NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A Platform Independent Application
of
Lux Illumination Prediction Algorithms

by

Michael Theodore Lester

June 1994

Thesis Advisor: Douglas J. Fouts

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REFORT Approved for public release, distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School — 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION — 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. — PROJECT NO. — TASK NO. — WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

A Platform Independent Application of Lux Illumination Prediction Algorithms

12. PERSONAL AUTHOR(S)

Lester, Michael Theodore

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year,Month,Day)	15. PAGE COUNT
Masters Thesis	FROM _____ TO_____	June 1994	126

16. SUPPLEMENTARY NOTATION

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
			Illumination, NVG, Illumination Prediction

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Naval Aviators require prior knowledge of the time and location of astronomical phenomena in order to properly plan and execute combat and training operations during the hours of darkness using Night Vision Devices (NVD's). This thesis presents a computer application of illumination prediction algorithms which predict the time of selected astonomical phenomena. This computer program is platform independent (given the proper libraries), event-driven, object-oriented, and utilizes a Graphical User Interface (GUI). Using this application, operators in the field will be able to determine the time of selected phenomena and the quantity of illumination, measured in Lux, for a given time and date.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
[X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) — 22c. OFFICE SYMBOL
Douglas J. Fouts	408-656-2852 — EC/FS

Approved for public release; distribution unlimited.

A Platform Independent Application
of
Lux Illumination Prediction Algorithms

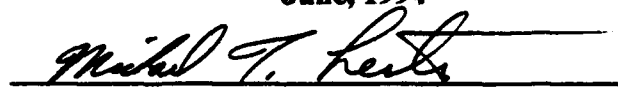
by

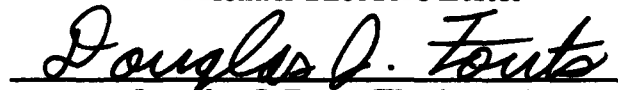
Michael Theodore Lester
Captain, United States Marine Corps
B.S., United States Naval Academy, 1985

Submitted in partial fulfillment of the
requirements for the degree of

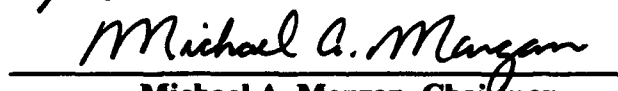
MASTER OF SCIENCE IN ELECTRICAL ENGINEERING
from the
NAVAL POSTGRADUATE SCHOOL

June, 1994

Author: 
Michael Theodore Lester

Approved by: 
Douglas J. Fouts, Thesis Advisor


Raymond F. Bernstein Jr., Second Reader


Michael A. Morgan, Chairman,
Department of Electrical and Computer Engineering

Abstract

Naval Aviators require knowledge of the time and location of astronomical phenomena in order to properly plan and execute combat and training operations during the hours of darkness using Night Vision Devices (NVD's). This thesis presents a computer application of illumination prediction algorithms which predict the time of selected astonomical phenomena. This computer program is platform independent (given the proper libraries), event-driven, object-oriented, and utilizes a Graphical User Interface (GUI). Using this application, operators in the field will be able to determine the time of selected phenomena and the quantity of illumination, measured in Lux, for a given time and date.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

Combat operations are increasingly conducted during the hours of darkness. Operations at night give a decided tactical advantage to the technologically advanced force possessing night vision devices (NVD's). Helmet mounted NVD's used by aviators are limited, however, to the simple amplification of ambient light.[1] Operations involving NVD's must be planned to coincide with appropriate illumination. Since NVD's are passive, i.e. they merely magnify available light, they require a minimum amount of illumination to operate. Additionally, since they magnify light on the order of 10,000 times (AN/PVS5: X10,000 gain, AN/AVS6: X25,000 gain)[1] , there can not be too much illumination or the NVD's will reach saturation and shutdown to protect their circuitry. Battlefield planners require a method for determining the time and location of astronomical phenomena such as sun rise, sun set, moon rise, moon set, and for determining the amount of light available from these phenomena. Previously, this information was available in tabular form in the Nautical Almanac or from a computer program called LITELEVL.

LITELEVL is written in the ubiquitous GWBASIC. It is completely text based and runs only on MSDOS compatible personal computers.[2] LITELEVL is not optimized for any parameters and thus has a response time of many seconds for a single line of a planning calendar output.

B. OBJECTIVES

This thesis will produce an improved application of the tested illumination prediction algorithms. This application, called MOONLITE to differentiate it from its predecessor, has the following properties:

- Platform independent. With proper libraries the source code may be compiled to run under Microsoft Windows™, Microsoft Windows NT™, IBM OS/2™, DOS™ (graphics mode), DOS™ (text mode), UNIX Motif, and (in the future) Apple Macintosh™.
- Graphical User interface featuring pull down menus and point-and-shoot dialog boxes.
- Event-driven architecture.
- Object-oriented design for ease of maintenance and upgrading.
- Ability to store multiple geographic locations for future analysis.
- Increased speed.
- Increased accuracy.

C. SUMMARY OF THESIS

1. Night Imaging Considerations

This section investigates the basic concepts of illumination and illuminance. It presents a short tutorial on the nature of light, the movement of the heavenly bodies, and the effect of meteorology on local illuminance. It also examines the effects of altitude, geography, and human interpretation on the accuracy of predicted astronomical events.

2. Prediction Algorithms

Two separate algorithms are used in MOONLITE. They are referred to as the "fast" algorithm and the "accurate" algorithm. This chapter examines both algorithms. It reviews their relative strengths and weaknesses and briefly explores their coding.

3. Testing and Validation

Prior to its release to the subordinate units of the Department of Defense, MOONLITE must be tested and validated by the United States Naval Observatory.

Initial testing was accomplished during coding. This chapter briefly introduces the benchmarks by which the data is judged. Rigorous testing will be accomplished at the Naval Observatory.

4. User Interface

The user interface is perhaps the most important part of a program. Regardless of the accuracy or efficiency of the underlying code, the user will either use or not use a program dependent upon the user interface. A great deal of time and effort was devoted to making MOONLITE's user interface intuitive, friendly, and efficient. This chapter examines the nuances of the user interface, detailing design considerations and decisions where necessary.

5. Coding Considerations

The coding of a program determines its accuracy and its efficiency. This chapter examines the general coding of the program. Class structures and data structures are examined in this chapter. Coding decisions and concerns are addressed, as are logic and program flow.

6. Future Work Needed / Upgrades

This chapter outlines future work which will be accomplished by the author at the next duty station. In addition, ideas for future enhancements are discussed.

7. User's Manual (Appendix)

This appendix is designed to serve as a user's manual for MOONLITE. It may be removed from the attached material and distributed with MOONLITE to the end user. It is designed to be concise enough to allow a person with limited technical background to

install and use MOONLITE. It describes the input and output requirements in addition to presenting the reader with an easy to follow tutorial for anticipated actions.

II. NIGHT IMAGING CONSIDERATIONS

A. THE NATURE OF LIGHT

Light, as we will use the term here, is the portion of the electromagnetic spectrum which is visible to the human eye. The electromagnetic spectrum spans all frequencies from sub-aural (less than 20 Hz) to cosmic rays (10^{22} Hz) and beyond. Electromagnetic energy with a wavelength between 400 and 700 nanometers is visible to humans. We call this range the visible spectrum. Immediately below the human threshold of vision is the near infrared region.[1]

Helmet mounted night vision devices used by aviators intensify available light. More specifically, they intensify the light which is reflected from an object. The amount of light reflected from an object is called luminance. An object's luminance is a function of how much light is striking the object, the illuminance, and the reflectivity of the object. With the same illuminance, a light object such as snow will have a greater luminance than a dark object such as an asphalt road. Prediction of night vision device's efficiency is confined to the prediction of the illuminance of all objects in a certain area regardless of their reflectivity and resultant luminance. Illuminance is expressed in Lumens per square meter, or Lux. One Lux is equal to 0.0929 foot-candles. Table 1 shows the relative illuminance of various sky conditions.[1]

TABLE 1. ILLUMINANCE LEVELS OF VARIOUS SKY CONDITIONS

Sky Condition	Approx. Illuminance (Lux)
Direct Sunlight	$1 - 1.3 \times 10^5$
Full Daylight (not direct)	$1 - 2 \times 10^4$
Overcast Day	10^3
Very Dark Day	10^2
Twilight	10
Deep Twilight	1
Full Moon	10^{-1}
Quarter Moon	10^{-2}
Moonless, Clear Night Sky	10^{-3}
Moonless, Overcast Night Sky	10^{-4}

Solar light, light emanating from stars, the moon, and other solar phenomena, is principally comprised of wavelengths outside the visible spectrum. For this reason, Night Imaging Devices are designed with their principle sensitivity outside of the visible spectrum and more into the near infrared spectrum. Figure 1 illustrates the relative wavelengths of human visible spectrum with that of the night sky and two modern night vision devices.[1]

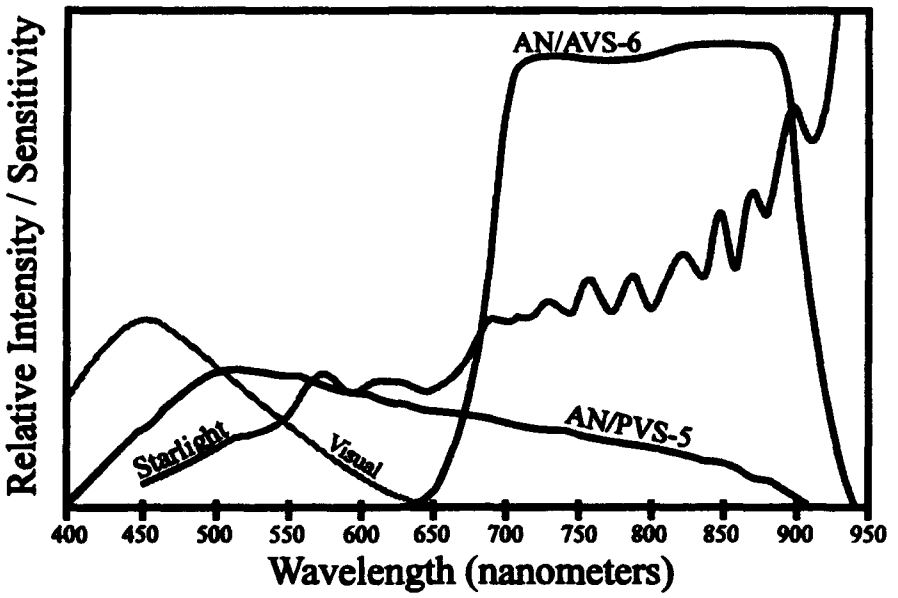


Figure 1. Comparison of visible spectrum versus night vision devices

The third generation AN-AVS6 Night Vision Goggles (NVG) are sensitive in an area well outside of the human visual spectrum. This greatly enhances their ability to amplify ambient light at night, but it hinders our ability to predict their efficiency as discussed in the next section.

B. PROBLEMS WITH ILLUMINANCE PREDICTION

1. Spectrum Selection

Modern illumination algorithms are designed to predict illuminance, the amount of visible light present. There have been no definitive studies to date on the amount of light presented at the surface of the earth during the hours of darkness in a spectrum other than that of the visible spectrum. In practice, the United States Military assumes that the illuminance in the visible spectrum is directly proportional to the amount of light in the near infrared spectrum.[3]

2. Atmospheric Attenuation

Since all light reaching the surface of the earth from the cosmos must pass through the atmosphere, the composition of the atmosphere has a measurable effect on the attenuation of that light.

The atmosphere is the most dense at the surface of the earth. As one increases in altitude the density of the atmosphere decreases. With this decrease in density, the attenuation effect of the atmosphere is also decreased. For computational purposes, the atmosphere is considered homogenous to an altitude of 8.46 kilometers. Using this simplification does not significantly alter the results of illuminance prediction. [4]

3. Altitude Discrepancy

The computation of astronomical phenomena is complicated at altitudes non-coincident with the surface of the earth. At high altitudes the atmosphere does not attenuate sunlight or moonlight to the same degree to which it does at sea level. The apparent rise and set of the heavenly bodies is offset at altitude. A person on the surface of the earth may observe sun set at the same time a person in a jet at 30,000 feet can still view the entire disc of the sun.

Generally accepted phenomena such as civil and nautical twilight are also offset at altitude. Civil and nautical twilight are defined as the time at which the sun is six and twelve degrees, respectively, below the horizon. As with sun set, there will be more light at altitude during twilight than their would be on the surface of the earth.[3]

4. Geographic Limits

Although geographic limits may seem trivial, they still affect the prediction of astronomical phenomena. Astronomical phenomena is predicted relative to a plane which is assumed to be tangent to the surface of the earth at the observer's latitude and longitude

at sea level. This assumption obviates the problems of a person standing on a mountain or in a valley and thus observing astronomical phenomena at a different time.

Illuminance is predicted assuming that the earth is visible to the source of illumination at the desired moment in time. For example, if a full moon has just risen and is currently only ten degrees above the horizon, a person on one side of a mountain will be fully illuminated. A person on the other side of the mountain would still be in darkness. Shadowing and obscuration is not addressed by the prediction algorithms. It is incumbent upon the user of the algorithms to understand the limitations of the predictions.

5. Meteorology

Cloud formations, fog, smog, and other obscurants in the sky will attenuate the amount of light impinging upon the earth from celestial sources. It is possible to account for this attenuation in a computer program, but it requires the end user to determine the quantity of attenuation. Since most end users will not have equipment capable of measuring cloud density, MOONLITE does not allow the user to apply an attenuation scaling factor. Aviators must use the experience gained during their NVG qualification to determine the amount of NVG degradation due to meteorology.[1]

6. Illuminance

Illuminance, as described above, refers to the amount of light being shed on an object. MOONLITE predicts the amount of light available from astronomical sources. An NVG user, though, may have artificial light available from a nearby city or town, or from battle field illumination, burning oil wells, etc. It is impossible for a predictive program to consider the myriad of artificial sources of light that might be present. Again,

the NVG user must use his or her experience to determine the amount of light available for NVG use.

III. ALGORITHMS

A. FAST VERSUS ACCURATE

MOONLITE uses two different sets of algorithms for determining celestial phenomena. The fist set of algorithms is referred to as the fast algorithms and are iterative in nature. These algorithms are found in the United States Naval Observatory Circular No. 171. The fast algorithm's most glaring shortcoming is that it diverges at latitudes above 60 degrees north or south.[5] This divergence can manifest itself as a complete miss of a phenomena such as sun rise. A more subtle error, however, could occur where the algorithm produced output that although flawed, appeared to be correct. The fast algorithms are not used for prediction above 60 degrees north or south latitude.

The other set of algorithms is referred to as the accurate algorithms. The accurate algorithms are currently being perfected at the United States Naval Observatory. These algorithms are interpolative in nature and will produce accurate output at any location on the earth. There is a trade off in processing time between the two algorithms. The accurate algorithms take substantially longer to compute an event than the fast algorithms. [3]. Although an advance copy of the accurate algorithm has been obtained prior to the completion of this thesis, they have not been implemented due to current instabilities which are being corrected by the Naval Observatory. MOONLITE was written to facilitate the accurate algorithms as soon as they are available. The remainder of this thesis will therefore deal with the fast algorithms.

B. ACCURACY

The design goal of the fast algorithm was to identify a phenomena within 0.5 degrees of its actual azimuth and altitude. The maximum temporal error, assuming a 0.5 degree error in placement, would therefore be two minutes. The temporal values are relative to

the mean time of the selected time zone. Each time zone (with variations for political and geographic anomalies) is fifteen degrees wide. Phenomena are computed using the time in the center of the time zone. An observer can expect a divergence from the predicted times relative to their distance from the center of the selected time zone.[3]

Rounding of numbers may cause discrepancies larger than the target goals in some cases. Consequently, the last digit of angles and times should be considered uncertain. [3]

Illuminance is given in Lux, and should be accurate to one or two digits. Due to local conditions (artificial light, meteorology, etc.) the calculated illuminance may differ from the actual illuminance by a factor of 10 or more.[3]

The Moon's apparent phase is independent of Earth's atmosphere, but approximations in the equations for calculating it may produce errors of one or two units in the computed quantity.[2]

MOONLITE was coded with all variables and constants defined as "double", thus internal computer "accuracy" is carried out to 64 bits using the IEEE real standard. This format allows representable numbers between -2,147,483,648 and 2,147,483,647.[A]

C. ALGORITHM OVERVIEW

1. General

The fast algorithm is presented in reference [3]. The algorithm is presented in BASIC and FORTRAN. In order to make MOONLITE platform independent, it was necessary to code MOONLITE in C++. Originally, compiling the FORTRAN code and linking it into a C++ user interface was contemplated, but this approach was rejected due to possible incompatibilities in multiple platform object code.

The FORTRAN and BASIC code was converted to C++ code for inclusion in MOONLITE. To aid in maintenance and debugging, the algorithms are designed as a separate class of MOONLITE named *fast_algorithm*. The C++ code for fast_algorithm may be found in Appendix A.

2. Coding Considerations

If one examines the original BASIC code and the MOONLITE code for the fast_algorithm class, one will notice that the MOONLITE fast_algorithm class does not use subroutines. This is contrary to modern modular programming technique, yet seemed advisable for MOONLITE. Using passive profiling techniques, the fast_algorithm was identified as a computationally intensive, and thus time consuming, component of the MOONLITE program. Since the algorithm is somewhat linear in flow, it was coded to minimize loop iterations and subroutine calls, and thus maximize runtime efficiency.[7]

By removing the subroutines and placing them "in-line", the code increased in length from 225 statements to 233, but it also removed 49 GOTO statements and 20 GOSUB statements.

Each GOSUB statement would cause a context switch with attendant overhead. The GOTO statement in GWBASIC, the language MOONLITE's predecessor was written in, is implemented by a linear search algorithm. MOONLITE uses structured programming to remove the GOTO statements. This structured technique allows the compiler to place a hard coded jump address in the machine code versus an iterative algorithm which takes multiple lines of machine code to implement.

Where possible, code was hand optimized while converting from BASIC to C++. For example, in the original BASIC code are the following lines:

```
340 FOR L = 1 TO 4                                    (1)
350 ON L GOTO 370, 650, 650, 360
```

```
360 C = 347.81
370 M = .5 + DT
```

This was replaced with:

```
for (int L = 1; L <= 4; L++){                                    (2)
if (L==4) c = 347.81;
if ((L==1) || (L==4)) {
m = 0.5 + DT;
:
:
}
//code for BASIC line 650 goes here
```

This rewriting of the BASIC code removes four GOTO statements and replaces them with two IF statements. Since the GOTO statements cause an iterative loop of many cycles whereas the IF statements do not, the latter code is markedly faster.

IV. TESTING AND VALIDATION

A. TESTING

1. Preliminary

During the translation of the fast algorithms from reference BASIC and FORTRAN to the C++ code of MOONLITE, constant testing was performed to ensure parallel results. For two randomly selected cases, the BASIC code and C++ code was stepped and compared line by line to ensure matching results after each statement.

Results between MOONLITE and LITELEVL are not exact. Discrepancies have been traced primarily to the way the two languages handle the trigonometric functions. Discrepancies manifest themselves as a difference in time of one minute or less. During initial testing, when a discrepancy was found between the BASIC handling of a SIN function and the C++ handling of the function, the calculation was run on an HP48SX calculator and on MATLAB V4.0 to compare results. In all cases the results of the HP calculator and of MATLAB matched the results of the MOONLITE C++ code. Thus, the MOONLITE C++ code is considered to be more accurate than the original BASIC code.

Reference [5], table A, presents test cases for program certification. MOONLITE was tested locally against these standards and found to be within one minute of time and within 0.5 degrees to all parameters. This is within design specification, and thus shows that MOONLITE's C++ code is functioning properly.

2. Final

Prior to release, MOONLITE will be tested against the United States Naval Observatory test suite. The Naval Observatory has determined a number of test conditions which will test boundary conditions in the program. These boundary conditions are the most ill-conditioned points the program can be expected to handle.[3]

B. VALIDATION

Upon completion, MOONLITE will be tested at the Unites States Naval Observatory by the Astronomical Applications Division. After successfully completing their tests, the program will be validated by the Observatory for use by the Department of Defense.

V. USER INTERFACE

A. GENERAL

There were four main criteria envisioned for the user interface. They were:

- Must be relatively intuitive
- Must be graphical
- Must be transportable
- Must minimize input errors

To ensure that these criteria were met, MOONLITE was developed in Microsoft Windows using Borland C++ version 4.0. Standard Windows conventions were adhered to, for example, the upper left hand corner of a window contains a control box which will allow the user to minimize, maximize, move, or close the window.[8]

B. MENUS

All menus are standard pull-down menus allowing the user to select an option by clicking on that option with the mouse. Menus were designed as "sticky" menus, i.e. the user does not have to hold the mouse button down to keep the menu active. Menus may be accessed without a mouse by pressing the ALT key and the first letter of the menu choice. For example, the user may activate the FILE menu by pressing ALT_F.

MOONLITE's menus are only one level deep. This was a conscious design choice to ensure the most simplistic and thus most easily understood user interface.

C. DIALOG BOXES

Almost every menu choice leads to a dialog box. Each dialog box contains a HELP button which will activate context sensitive help for that dialog. Currently, context

sensitive help is disabled during research into the portability of Microsoft Windows help resources.

VI. CODING CONSIDERATIONS

A. LANGUAGE CHOICE

MOONLITE was written in Borland C++ version 4.0. This language was chosen for a number of reasons. First, C++ is the most ubiquitous object-oriented language in use today. MOONLITE was written using object-oriented techniques in order to facilitate future maintenance and upgradability. Second, NAVAIR has expressed interest in interfacing MOONLITE with TAMPS (Tactical Air Mission Planning System) and TAMPS is written in C++. Third, multiple platform libraries were available for C++. Borland's product was chosen as development suite because of its enhanced development and debugging tools.

B. LIBRARIES

ZAPP libraries were purchased from Inmark corporation for the development of MOONLITE. Inmark currently supplies libraries for Microsoft Windows™, Microsoft Windows NT™, Microsoft DOS™ (graphics mode), Microsoft DOS™ (text mode), IBM OS/2™, UNIX MOTIF™, and in the near future for Apple Macintosh™. Using these libraries, it is possible to recompile source code written to take advantage of the libraries for a different platform with out re-coding. This saves development cost and presents the end user with a consistent interface across multiple platforms. Currently, MOONLITE is compiled only for Microsoft Windows™.

C. PROGRAM STRUCTURE

1. Control Hierarchy

MOONLITE is an event driven program. Event driven programs perform no function until prompted by an event. MOONLITE, once running, simply waits for the

user to initiate an event. The most common event will be a menu choice, but it could also be a window resize, move, or termination.

The top level of the MOONLITE program is the event handler. The event handler fields all events and then instantiates and sends a message to client classes as needed. A hierarchy of events is shown in Figure 2.

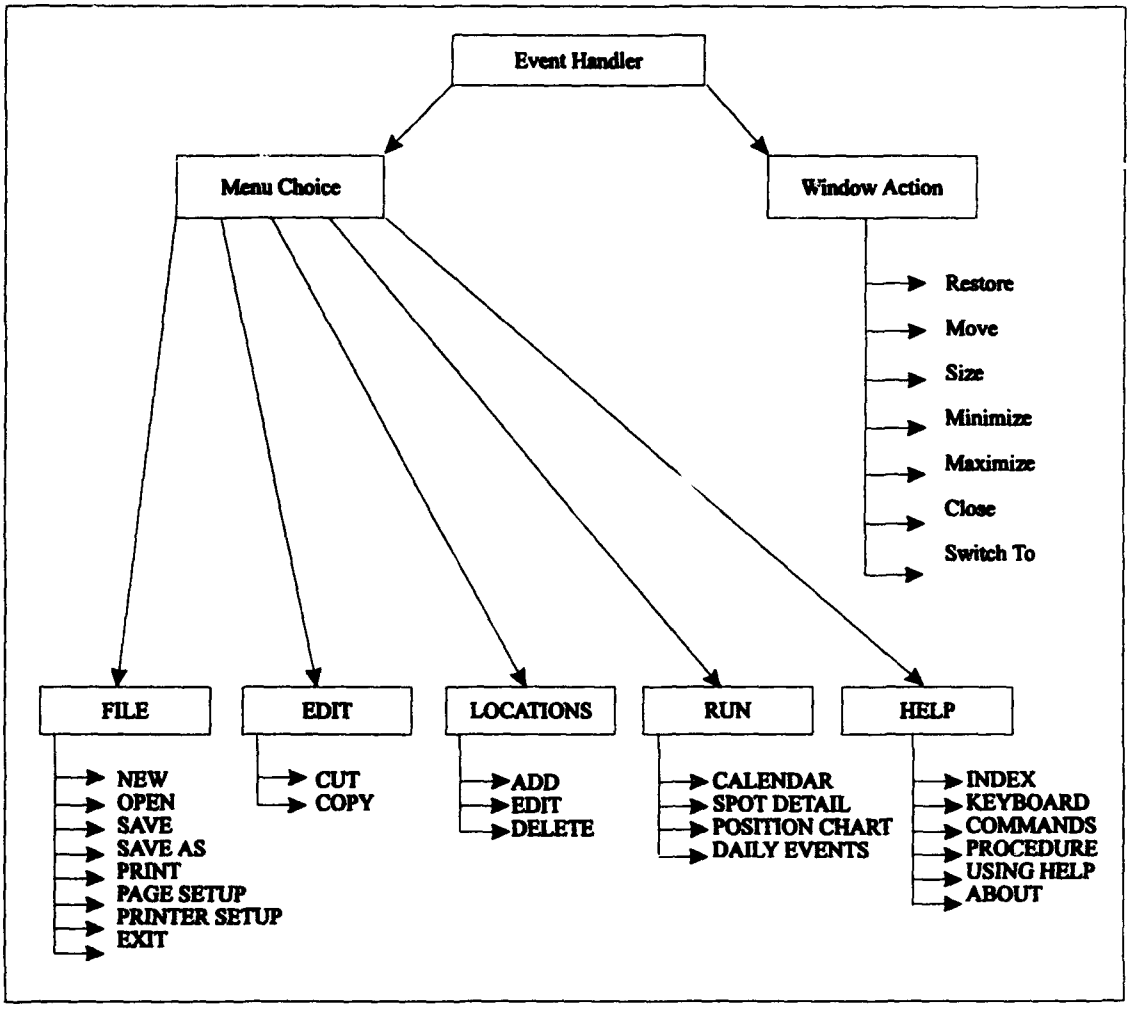


Figure 2. Event Handling Hierarchy

The single level menuing of MOONLITE is evident from Figure 2. Each menu option is modal in MOONLITE, i.e. another option cannot be chosen until the user is

finished with the current option. Rewriting MOONLITE to handle MDI (Multiple Document Interface) may be an option for future enhancement.

It is important that programs written to operate in a multitasking environment, such and Microsoft Windows or UNIX, be event-driven. An event-driven program will "sleep" when it is not fielding events. This allows maximum processor utilization since the program is not executing a "busy-loop" or polling for user input and thus allows other processes to execute while it is waiting for an event.[7]

2. Classes

As stated earlier, MOONLITE is written using object oriented techniques. Object oriented programming consists of encapsulating member functions and variables in a class, the instantiation of which is an object. The basic class hierarchy of MOONLITE is shown in Figure 3.

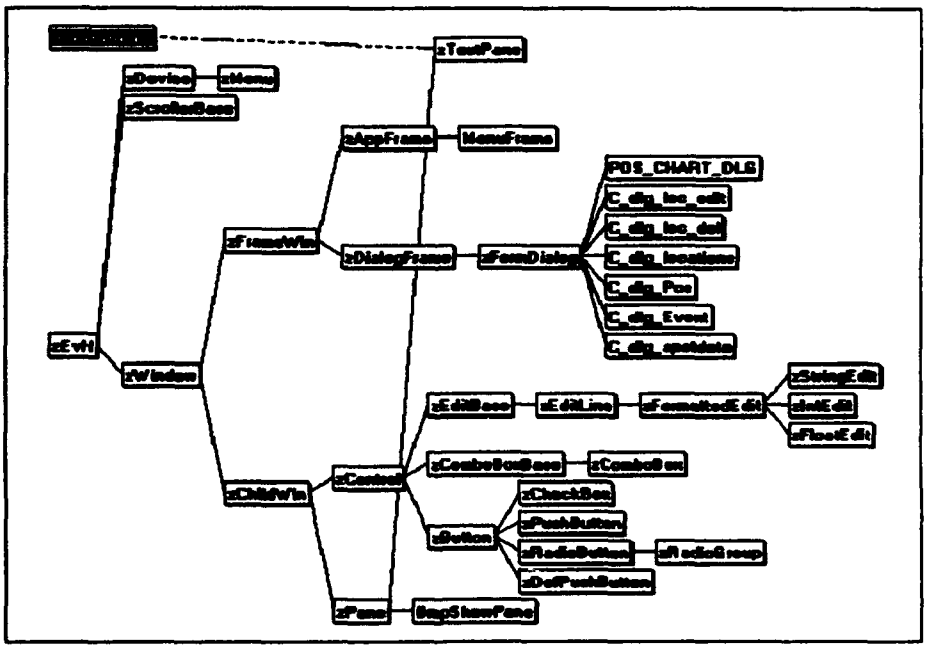


Figure 3. Class hierarchy of derived classes

The base class from which most other classes are derived is zEvH, the event handler. Derived from zEvh is zWindow and zFrameWin. ZFrameWin is a class which when instantiated and given focus presents a standard windows frame on the monitor. This frame can not hold text or graphics, but it can hold other windows which do hold text and graphics. ZFrameWin's purpose is to create a standard Windows window capable of resizing, moving, minimizing, maximizing, and terminating. [8]

Derived from zFrameWin is zAppFrame and MenuFrame. ZAppFrame is a special zFrameWin which is designed to be the topmost window of an actual application such as MOONLITE. Code segment (3) illustrates the code used to instantiate the upper level application window for MOONLITE.

```
MenuFrame *mainWnd=new MenuFrame(0,new zSizer(10,10,625,520),zSTDFRAME,"MOONLITE");      (3)
```

Code segment (3) instantiates a new instance of a class of MenuFrame with an attendant pointed called mainWnd. In its constructor, it sets a default window size using the class zSizer. In this instance the window's upper left hand corner starts at pixel 10,10 and ends at pixel 625,520. This size is aesthetically pleasing for a standard 1024 X 768 pixel display.

After instantiating the main application window the menu items are added. The menu itself was created using the resource workshop of Borland C++. Using the resource workshop, a description of the desired menu is created then passed to the main application window for display. Code segment (4) shows the only MOONLITE line of code needed to attach a menu to the parent application.

```
menu(new zMenu(this, zResId(MENU_MAIN)));                                              (4)
```

Code segment (4) instantiates a new zMenu and passes an identification number to the constructor. This identification number is assigned to the menu resource and is a method of uniquely identifying resources. A portion of the menu description is shown in code segment (5). One can see that the menu description includes the displayed title of the menu item, its parent menu item, a control key keyboard shortcut if available, and an identity number to be passed to a function should that menu item be called.

```
POPUP "&File"                                                                    (5)
{
 MENUITEM "&New", ID_MENU_FILENEW
 MENUITEM "&Open...", ID_MENU_FILEOPEN
 MENUITEM "&Save", ID_MENU_FILESAVE
 MENUITEM "Save &as...", ID_MENU_FILESAVEAS
 MENUITEM SEPARATOR
 MENUITEM "&Print...", ID_MENU_FILEPRINT, GRAYED
 MENUITEM "Page se&tup...", ID_MENU_FILEPAGESETUP, GRAYED
 MENUITEM "P&rinter setup...", ID_MENU_FILEPRINTERSETUP, GRAYED
 MENUITEM SEPARATOR
 MENUITEM "E&xit", ID_MENU_FILEEXIT
}
```

In order to have our desired function called when menu item is selected, it is necessary to override the default event handler for each menu item. Code segment (6) illustrates how the event handler is overridden.

```
menu()->setCommand(this,(CommandProc)&MenuFrame::doExit,ID_MENU_FILEEXIT);                    (6)
menu()->setCommand(this,(CommandProc)&MenuFrame::AddLocation,ID_MENU_LOCS_ADD);
menu()->setCommand(this,(CommandProc)&MenuFrame::EditLocation,ID_MENU_LOCS_EDIT);
menu()->setCommand(this,(CommandProc)&MenuFrame::DeleteLocation,ID_MENU_LOCS_DELETE);
menu()->setCommand(this,(CommandProc)&MenuFrame::FileSave,ID_MENU_FILESAVE);
menu()->setCommand(this,(CommandProc)&MenuFrame::FileOpen,ID_MENU_FILEOPEN);
menu()->setCommand(this,(CommandProc)&MenuFrame::FileNew,ID_MENU_FILENEW);
menu()->setCommand(this,(CommandProc)&MenuFrame::SpotData,ID_MENU_SPOTDATA);
menu()->setCommand(this,(CommandProc)&MenuFrame::Event,ID_MENU_DAILYEVENTS);
menu()->setCommand(this,(CommandProc)&MenuFrame::Position,ID_MENU_POSITION);
```

Here again we see an identifying number passed to each function in the form of a descriptive name. The file *defines.h* contains a list of all identifiers used in MOONLITE and their corresponding numeric identifier. Numeric identifiers are arbitrarily chosen. In MOONLITE each logical grouping is given its own hundreds digit identifier with

member items being numbered sequentially. Code Segment (7) is an example of some identifiers from *defines.h*.

```
#define MENU_MAIN                100                    (7)
#define ID_MENU_POSITION         118
#define ID_MENU_FILENEW          101
#define ID_MENU_FILEOPEN         102
#define ID_MENU_FILESAVE         103
#define ID_MENU_FILESAVEAS       104
#define ID_MENU_FILEPRINT        105
#define ID_MENU_FILEPAGESETUP    106
#define ID_MENU_FILEPRINTERSETUP 107
#define ID_MENU_FILEEXIT         108
```

In C++ each define is substituted by its defined value at compile time. Thus ID_MENU_FILEEXIT is replaced at compile time by the number 108. Using "define" statements makes the code more readable and decreases the chances of assigning a wrong identifier to a function.

Each menu function instantiates a type of zFormDialog. MOONLITE handles all input and output through the use of dialog boxes. This technique allows the end user to size and move output windows on the monitor to suit their individual taste. There are eight dialog classes. Each dialog class begins with "C_dlg" to identify it as a class and a dialog. These classes are examined in the next section.

3. Program Flow

When a user selects a menu item, that item's corresponding class is instantiated and called. Focus is passed to that class (always a dialog) and retained until the dialog is closed and deleted. As an example of program flow, we will trace the flow of the menu selection RUN|SPOTDATA.

When a user initially runs MOONLITE, the program places itself in memory and attempts to read a file called *moonlite.dat*. If that file is not found, MOONLITE presents a warning message to the user telling them that the data file was not loaded. If the file was found, MOONLITE loads the locations stored in *moonlite.dat* into a linked list. This

linked list is passed to all of the dialogs from this point on in the program. Each dialog requires that the user select a location for which they would like output. Using a linked list allows a (theoretically) unlimited number of locations to be stored by MOONLITE.[10]

Once MOONLITE is initialized, it waits for the user to generate an event. In this case we will assume that the user points to the menu item RUN and selects the member item SPOTDATA. From code segment (6) we see that menu item ID_RUN_SPOTDATA is associated with event handler MenuFrame::Spotdata. MenuFrame::Spotdata is comprised of the following code:

```
MenuFrame::SpotData(zCommandEvt *ce)                                    (8)
    {
     routines *engine = new routines(ll);
     engine->spotdata();
     delete engine;
     return(1);
    }
```

MenuFrame::Spotdata creates a new object of type routines, and assigns a pointed named "engine" to point to that object. Notice that "ll" the linked list of locations was passed to the class "routines" as part of the constructor. Once the class is instantiated as an object, the object is sent the message "spotdata".

The code associated with routines::spotdata is shown in code segment (9).

```
routines::spotdata()                                                    (9)
       {
        C_dlg_spotdata *dlg_spot = new C_dlg_spotdata(ll, zResId(SPOT_DETAIL));
        delete dlg_spot;
       }
```

Here again, this object creates a new object of the type C_dlg_spotdata. This is our dialog and the location where MOONLITE will interact with the user. Since C_dlg_spotdata always accomplishes the same task, the bulk of the user task is included in the constructor of the class and is automatically executed.

The code for C_dlg_spotdata is too lengthy to list here, but it is included in Appendix B. C_dlg_spotdata creates a dialog box with the necessary controls to prompt the user for the required input. It also contains read only controls for the display of output.

Like the main application, the dialog box for C_dlg_spotdata is event driven. Once instantiated, it waits for the user to generate an event. Assuming that the user wishes to continue with the spotdata, he or she will (in any order) select a location, a date, and a time. Each time an event is generated, C_dlg_spotdata examines the event and takes action on it. If the user selects a location, the dialog updates its internal variables with the data from the selected location.

The dialog has built in error reduction. A user may only choose the options that are available. He or she may not input anything from the keyboard. For example, the months of the year are presented to the user in a pulldown selection box. The user may only choose one of the twelve months. Once the user selects one, the dialog will analyze the choice and change the remaining dialog boxes to reflect that choice. For example, if the user selects "January", the dialog will change the possible selections of the day to include 31. If the user selects "February", the dialog will check the year to see if it is a leap year and present the appropriate number of days for the user to select. This pseudo real-time error checking reduces the number of errors a user can create. If a user should select the 29th day of February and later change the year to one that was not a leap year, MOONLITE will clear the day selected and highlight the day control to indicate to the user that they must select a new day.

Once the user selects a location, MOONLITE will compute the desired data on the fly each time the user changes a control. MOONLITE accomplishes this by examining the complete data set of the dialog box each time the user generates an event. Actually,

MOONLITE waits until the user completes an event, but to the user that is invisible. If the dialog is complete, i.e. the minimum number of controls to produce meaningful results have been selected and properly activated, MOONLITE instantiates the algorithm and passes the data to the new instantiation. When the algorithm returns the data, MOONLITE passes that data to the dialog's read-only controls for display.

When the user selects the DONE button, MOONLITE backs out of each class calling its destructor and freeing any memory used by that class. At this point MOONLITE is again in its initial state and waits for another event.

By designing MOONLITE to always return to its initial state, memory requirements are kept to a minimum. MOONLITE does not "hold" any memory aside for data storage or for computations. All memory requests are dynamic with the current event. MOONLITE was written to reduce its memory requirements after every operation for several reasons. First, when operating in a multitasking environment, MOONLITE ensures a maximum amount of memory available to the other processes. Second, when operating in a single task environment such as DOS, it is possible to overlay different class instantiations and therefore allow MOONLITE to operate in a much smaller memory area. MOONLITE therefore has a minimum impact on multitasking systems such as Microsoft windows™. [7]

4. Calendar Optimization

One of the most used functions of LITELEVL, and thus of MOONLITE is the planning calendar. The planning calendar presents the user with a graphical representation of illuminance levels for an extended duration. A typical request for the calendar option is to print the illuminance levels for a specific location from sun set to sun rise every night for a six month period. Calculating the calendar is the most

processor intensive procedure MOONLITE must accomplish. The old LITELEVL program frequently took more than a minute to compute one line of the calendar. One line corresponds to one nights worth of data. Inspection of the LITELEVL code reveals that besides using time consuming GOTO and GOSUB statements, the program used a brute force method of computing the calendar line.

LITELEVL attacked a planning calendar line in the following fashion: First, it computed the daily events for the current day. Then it computed the illuminance every ten minutes from 1700 until 0800 the next day, 82 points total. It then computed the daily events for the next day.

MOONLITE uses a more dynamic approach to the problem. MOONLITE first calculates the daily events for the current day. This calculation results in the sun rise, sun set, moon rise, and moon set. Using this data, MOONLITE compares the sun set time to the moon rise time. If the moon rises after the sun sets, MOONLITE automatically fills in the calendar line with the appropriate symbols to show that either the sun was still above the horizon or that the moon had not yet risen. If the moon rises after midnight MOONLITE fills in the entire first half of the calendar line with the symbol for the moon having not yet risen. If the moon rises before midnight, MOONLITE fills in the symbol for moon not risen up to the time of moonrise. From this point until either moon set or sun rise, MOONLITE uses a modified binary search algorithm to determine the state of illuminance.

Aviators only wish to know four conditions on a planning calendar: the sun is above the horizon, the moon is below the horizon, the sun is below the horizon and the illuminance is below a threshold, or the sun is below the horizon and the illuminance is above a threshold. Once the moon has risen, MOONLITE computes the illuminance for the first period. MOONLITE allows the user to select the period between calculations.

LITELEVL only calculated at ten minute intervals. MOONLITE then jumps four blocks forward and calculates the illuminance. If the illuminance results in the same category as the illuminance four blocks prior, MOONLITE fills in the sandwiched intermediate blocks with the same symbol. If the illuminance results in a different category, MOONLITE jumps back two blocks and computes that illuminance symbol. If the symbol is the same as the first symbol, MOONLITE fills in the second block with the same symbol and computes the third block. If the symbol is different from the first symbol, MOONLITE computes the second block. MOONLITE then compares the second block to the fourth block. If they are the same, if fills in the third block. If they are different, if computes the third block.

Using this jump and evaluate method, similar to a binary search, MOONLITE dramatically reduces the amount of calculations needed for each line of the calendar. At a minimum, if the moon never rises, MOONLITE will make no illuminance calculations and two daily event calculations. On the average, MOONLITE appears to make approximately seven illuminance calculations per calendar line.

Since both LITELEVL and MOONLITE make two daily event calculations per calendar line, we can compare the illuminance calculations for a rough indication of output speedup between LITELEVL and MOONLITE. LITELEVL always makes 82 calculations per line. Thus MOONLITE is 82/7 as fast as LITELEVL, or accomplishes a speedup of a little over eleven times the output speed of LITELEVL.

MOONLITE also enjoys a performance increase from the use of Borland's C++. The C++ language is an extremely terse, compact, and efficient language. Borland's C++ compiler uses a number of optimization techniques such as common subexpression elimination, copy propagation, and invariant code motion to further increase the runtime efficiency of the code. The final version of MOONLITE will be available in processor

specific versions which will optimize MOONLITE for use on that processor. The processor specific versions will take advantage of optimization techniques such as strength reduction and branch offset optimization. [5]

Figure 5 shows a single line from the planning calendar for a location at 36 degrees, 35.2 minutes north, 121 degrees, 50.6 minutes west on the 4th of November, 1993 with an offset of eight hours from Greenwich mean time. Figure 5 illustrates the order in which MOONLITE would analyze the example calendar line. Even though MOONLITE uses ten calculations on this line, it is still far faster than LITELEVL which calculates 82 points.

The jump MOONLITE makes between calculations nine and ten deserves explanation. MOONLITE compares every point prior to midnight to the point at midnight. As soon as there is a match, MOONLITE fills in the symbols and jumps to the first block after midnight. MOONLITE compares every point after midnight to the last point prior to sun rise. Again, as soon as there is a match, MOONLITE fills in the symbols between the two points. In our example, the symbol just after midnight and the symbol just prior to sun rise match, therefore MOONLITE fills and finishes.

From this example one can see that MOONLITE must make more calculations when the moon is most dynamic and fewer calculations when the moon is fairly static. The moon is most dynamic relative to the higher latitudes. Above sixty degrees north or south latitude, the moon could rise and set many times during a night. Indeed, at the pole, the moon could "hover" right on the verge of rising and setting.[9] Because of the ill-conditioned behavior of the moon at high latitudes, MOONLITE only uses this pseudo binary search routine with the fast algorithms at latitudes less than sixty degrees.

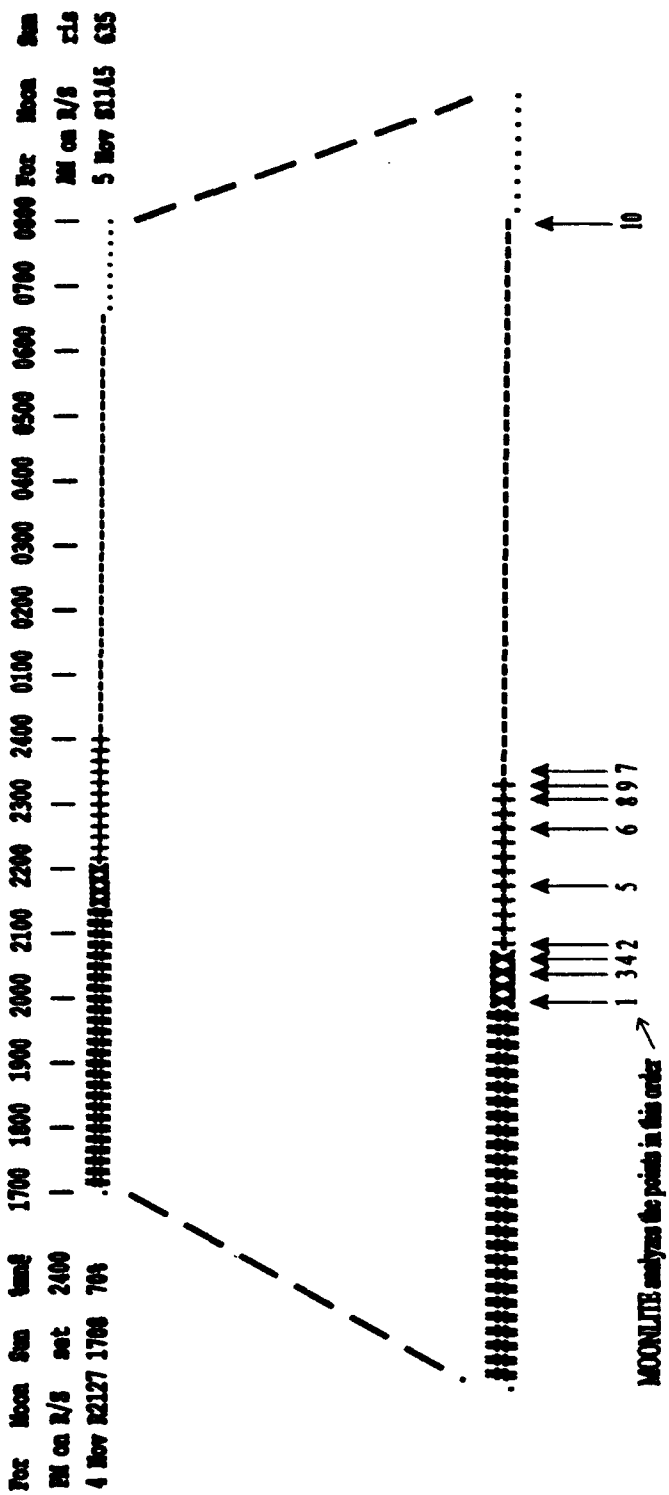


Figure 7. An example of MOONLITE's pseudo binary search algorithm

VII. FUTURE WORK NEEDED / UPGRADES

A. FUTURE WORK NEEDED

MOONLITE is not fully complete at this time. Although functional, some menu options and button choices are not activated. Per an agreement with the Naval Air Warfare Center in Warminster, Pennsylvania, the author will continue to work on the MOONLITE project at his next duty station to complete the MOONLITE project.

The following items need to be completed:

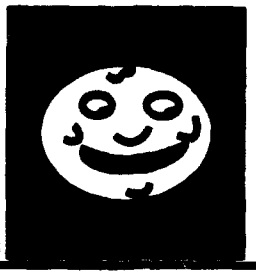
- Write PAGE SETUP routines
- Write PRINTER SETUP routines
- Write PRINT function
- Activate EDIT|COPY to allow copying data to Windows clipboard
- Activate context sensitive help
- Implement accurate algorithms

With the exception of implementing the accurate algorithms, all of the items remaining to be completed are dependent upon cross-platform considerations. For example, the PAGE SETUP, PRINTER SETUP, and PRINT routines can all be implemented using a single windows call each in Microsoft Windows™. When MOONLITE is ported to Microsoft DOS™, however, these calls will not be available. The author is awaiting receipt of DOS libraries prior to writing these routines to ensure their ability to compile both under Microsoft Windows™ and Microsoft DOS™.

B. FUTURE UPGRADES

During the design phase of MOONLITE, numerous Marine Corps and Navy pilots were queried regarding their desires in a program such as MOONLITE. From their responses, a list of future enhancements is proposed.

- Allow Dynamic Data Exchange from MOONLITE
- Add an option to print a graphical representation of moon azimuth and altitude versus time.
- Compile MOONLITE for use on the Apple Macintosh™.



Introduction

Welcome to **MOONLITE**!

MOONLITE is a computer program that predicts the normal daily celestial events such as sunrise, sunset, moonrise, moonset, and the phase of the moon. Additionally, **MOONLITE** calculates the amount of light being cast upon the surface of the earth from the sun, moon, and stars.

This initial release of **MOONLITE** is written to run within Microsoft Windowsä. Depending on the need for such versions, future versions of **MOONLITE** may run on Microsoft DOS™, on the Apple Macintosh™, and on UNIX-based workstations under MOTIF. Users of **MOONLITE** will experience a familiar interface regardless of which platform they use.

MOONLITE is distributed on a single 3½", 1.44 Megabyte diskette and requires Microsoft Windowsä version 3.0 or higher to run.

MOONLITE was developed at the United States Naval Postgraduate School in Monterey, California by Captain Michael T. Lester with advice and guidance from Doctor Douglas J. Fouts and Doctor Paul M. Janiczek.

Questions regarding the programming or operation of **MOONLITE**, reports of bugs (I hope not!), or suggestions for future enhancements may be sent via E-mail to "moonlite@ece.nps.navy.mil".

I have endeavered to make **MOONLITE** as user-friendly and efficient as possible and hope you enjoy its use.

Whats New!

MOONLITE is a completely new product!

The following is a list of **MOONLITE**'s new features:

- Runs under Microsoft Windows
- Graphical User Interface
- Stores locations for future use
- Optimized
- Variable time periods in Calendar and Position Chart

MOONLITE runs under Microsoft Windows. By writing **MOONLITE** to run under Windows we were able to take advantage of Window's Graphical User Interface (GUI) assuring you, the user, the most intuitive, easily used interface available.

MOONLITE's Graphical User Interface takes advantage of pull-down menus, dialog boxes, hot-key activation, and window sizing.

MOONLITE allows the user to store a virtually unlimited number of locations for future use. Multiple data files may also be used.

MOONLITE is optimized. **MOONLITE** is based on the same algorithms used by LITELEVL and SLAP, but they have been optimized and implemented in a way that ensures the fastest most accurate data available from these algorithms. **MOONLITE** was written using the most modern programming techniques which has resulted in a response time that is ten times faster than the old LITELEVL program.

Whats *Not* New!

MOONLITE uses the same algorithms as Litelevl and SLAP.

The output of the Position Chart and the Planning Calendar are unchanged. This was a conscious design decision. We believe that most users of MOONLITE are so familiar with the output of Litelevl that changing the format would be counter-productive.

Installation

General

MOONLITE is distributed on a single 3½", 1.44 Megabyte diskette. The disk contains the following four files:

1. MOONLITE.exe The MOONLITE program
2. MOONLITE.dat A data file of pre-loaded geographic locations
3. MOONLITE.doc A Winword copy of this document
4. Setup.exe The MOONLITE setup file

Installing MOONLITE on your Hard Drive

1. Start Microsoft Windows.
2. Place the MOONLITE diskette in the disk drive.
3. Open the "Main" Group by double clicking on the group entitled "Main" in the Program Manager.
4. Open the File Manager by clicking on the file manager icon in the Main group.
5. Click on the appropriate drive letter for your machine. On your machine, it may be drive A or drive B.
6. Double click on the file entitled "setup.exe" in the drive window.

Or,

1. Choose the menu option FILE | RUN from the Program manager
2. Type in "A:\setup.exe" or "B:\setup.exe" depending on which drive your MOONLITE diskette is in.

Setup

MOONLITE's setup program will perform the following tasks:

1. Create a directory on the C: drive called "MOONLITE"
2. Copy the files from the A: (or B:) drive to the newly created directory
3. Create a program group called MOONLITE
4. Place an icon for the MOONLITE program in the MOONLITE group

Running MOONLITE

From the Hard Drive

After installing MOONLITE using the Setup program as described above, MOONLITE may be run by double clicking on the MOONLITE icon.

From Diskette

MOONLITE may be run from a floppy diskette although loading time will be greatly reduced if the executable file, MOONLITE.exe, is placed on your hard drive.

To run MOONLITE from the diskette:

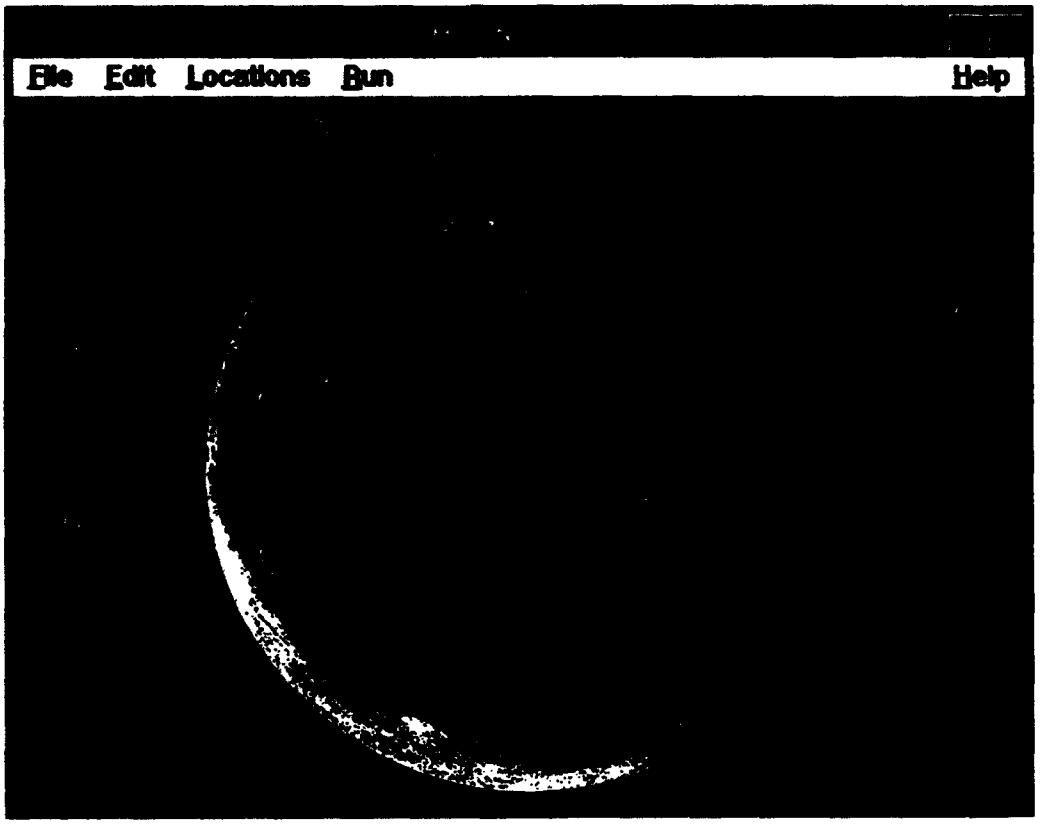
1. Start Microsoft Windows.
2. Place the MOONLITE diskette in the disk drive.
3. Open the "Main" Group by double clicking on the group entitled "Main" in the Program Manager.
4. Open the File Manager by clicking on the file manager icon in the Main group.
5. Click on the appropriate drive letter for your machine. On your machine, it may be drive A or drive B.
6. Double click on the file entitled "moonlite.exe" in the drive window.

Or,

1. Choose the menu option FILE | RUN from the Program manager
2. Type in "A:\moonlite.exe" or "B:\moonlite.exe" depending on which drive your MOONLITE diskette is in.

Tutorial

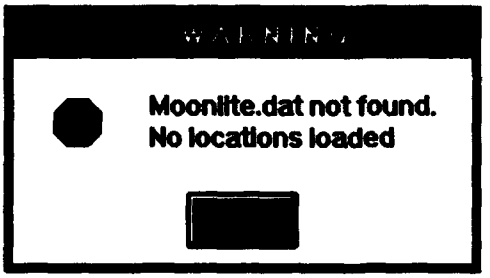
MOONLITE was designed to be as intuitive and user friendly as possible. When you first start MOONLITE you will be presented with the main MOONLITE screen seen below.



MOONLITE has five main menu choices: File, Edit, Locations, Run, and Help.

Unlike MOONLITE's predecessor, LITELEVL, MOONLITE allows the user to store geographic locations for future use. As a matter of fact, MOONLITE demands that the user load a location prior to use. This feature saves the user from looking up the latitude and longitude over and over every time they run the program. It also ensures that the user receives the data for the same latitude and longitude once they select a location.

When MOONLITE is first started, it looks for a file called "MOONLITE.dat". This file contains the geographic locations that have been previously stored in MOONLITE. There is theoretically no limit to the number of locations a user can store. The maximum number of locations a user can store is determined by the amount of memory the user has available. If MOONLITE can not find the data file, it warns the user that the data file is not found.



If you see this warning, it means that the file "moonlite.dat" was not located in the default directory nor in your current path. By clicking on the "OK" button, MOONLITE will continue to load and run.

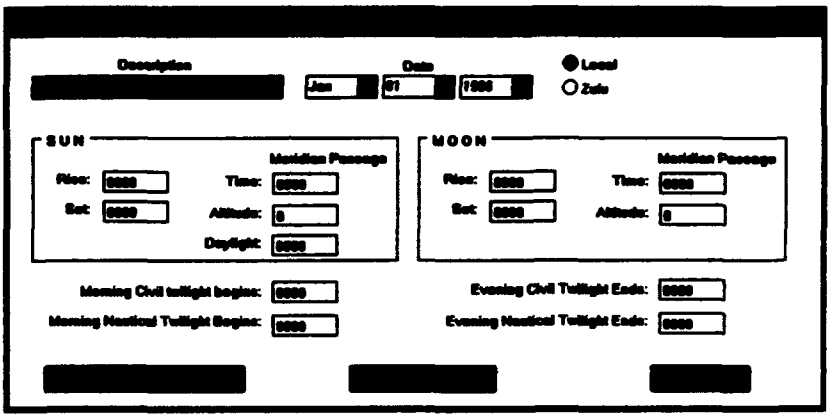
In order to allow the user to maintain geographic locations in logical groups, MOONLITE allows the use of of many data files although only one file may be in use at any time. By pulling down the FILE menu from the main MOONLITE screen, the user may choose to open a new data file or save a current data file.

For the time being, we will assume that the default data file "moonlite.dat" was loaded when MOONLITE was first started.

Since there are a number of locations pre-loaded with the MOONLITE program, we can proceed directly to the RUN menu to get our first results.

Clicking once on the RUN menu displays the output options that are available from MOONLITE. Lets find out when the Daily Events take place today. Click on the option "Daily Events".

MOONLITE now displays the dialog box for daily events. This dialog box is used both for you to tell MOONLITE what data you would like, and for MOONLITE to display that data for you.



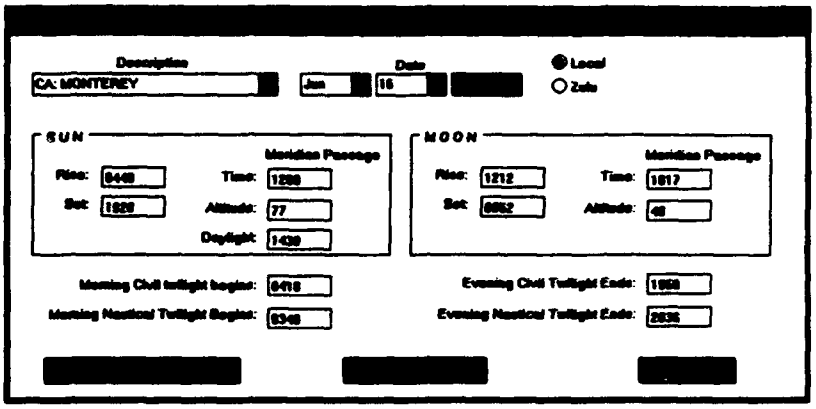
To determine the Daily Events, MOONLITE Needs to know the location and the date for which you desire data.

By clicking on the arrow to the right of the box entitled "Description", the one with "[Choose Location]" in the box, MOONLITE will display a list of the currently loaded locations. For this example, click on the location entitled "CA: Monterey". You can see that as soon as you made your selection, MOONLITE started giving you data. Right now, that data is for January 1, 1986 since that is the default date. By choosing the date the same way we chose the location, by clicking on the arrow on the right side of the combination box and selecting one of the options from the list, we can view data for any date within MOONLITE's operating range.

HINT: Combination boxes can be activated by clicking anywhere on the box, not just on the arrow on the right side.

For our example here, choose June 16, 1994. As you finish selecting the last parameter, MOONLITE has already updated the information and is displaying it in the lower half of the dialog box.

In this case, the Daily Events dialog looks like this:



Here in one easy to read dialog box we can see the location for which we desired data, the date for which we requested the data, and the data itself.

If you wish a print out of this data, you simply click once on the "Print" button on the bottom of the dialog box. For now, we want to explore some other parts of MOONLITE, so click on the "Done" button to return to the main MOONLITE screen.

Let's select the next option without the use of the mouse. You will notice that all of the menu options have one letter underlined. This letter indicates the "hot-key" which will activate the menu. By pressing down the "Alt" key and the letter that is underlined at the same time, we can activate the menu.

Press the "Alt" key on the keyboard and while continuing to hold it down, press the "R" key. Now let go of both keys.

You can see that the RUN menu was activated. You will also notice that each menu option inside of the RUN menu also has a hot-key associated with it. Since we already have the menu's attention, we don't need to use the Alt key this time, we only need to press the letter of the option we want. We have already looked at the Daily Events, so lets look at the Spot Detail this time. Since "S" is underlined in the "Spot Data" option, press the "S" key.

MOONLITE has responded by bringing us to the Spot Data dialog box. This dialog box is similar to the Daily Events dialog box, and we will tell MOONLITE what data we want in the same way.

Lets look at the spot data for Montery, CA on August, 21, 1994 at 9:00 O'clock in the morning.

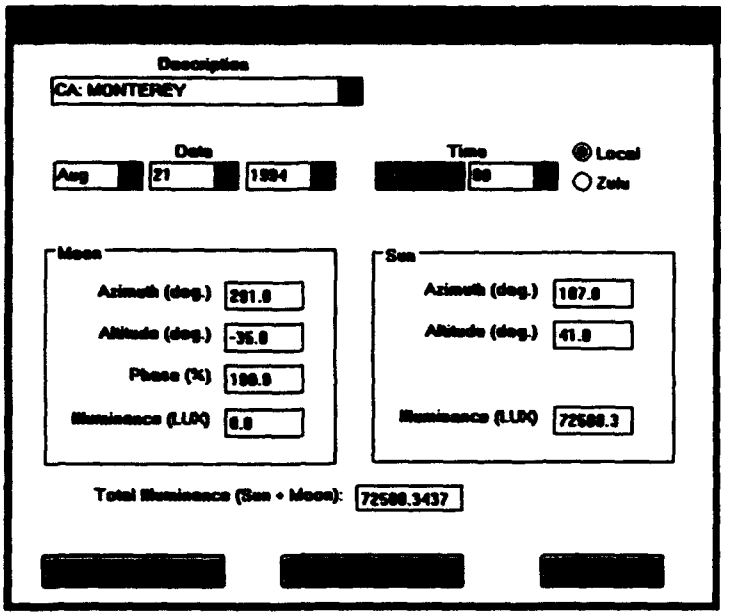
Choosing the location is done the same as it was in the Daily Events dialog box, so is choosing the month. When we try to choose the 21st day, however, we see that the selections only go to 16. We need to scroll the choices down to reach the other dates. We can do this by clicking anywhere below the position box on the scoll bar on the right side of the choices, or we can scroll one choice at a time by clicking on the down arrow on the bottom of the scroll bar.

But wait a minute! We weren't going to use the mouse this time, right? You can still enter all of the information. You will notice that the Location combination box is highlighted right now. That means that it is active and that keystrokes entered on the keyboard will affect that control. Since we want to look at "CA: Monterey", you can press "C" to have MOONLITE jump to the first choice that begins with "C". Since that isn't the choice we want, use the down arrow on the keyboard to scroll through the other options.

When "CA: Monterey" is highlighted, we can move to the next contol by pressing the tab key. When you pressed the tab key you notice that the focus shifts to the Month box. If you want to go back to the description box, press "alt-tab".

We enter the desired time in the same fashion.

After we are done, the Spot Detail dialog looks like this:



Just as in the Daily Events, we can press the "Print" key to get a printout of our data; we can select other dates, times, or locations; or we can press the "Done" key to end this event.

Go ahead and press the "Done" key. We have more exploring of MOONLITE to accomplish.

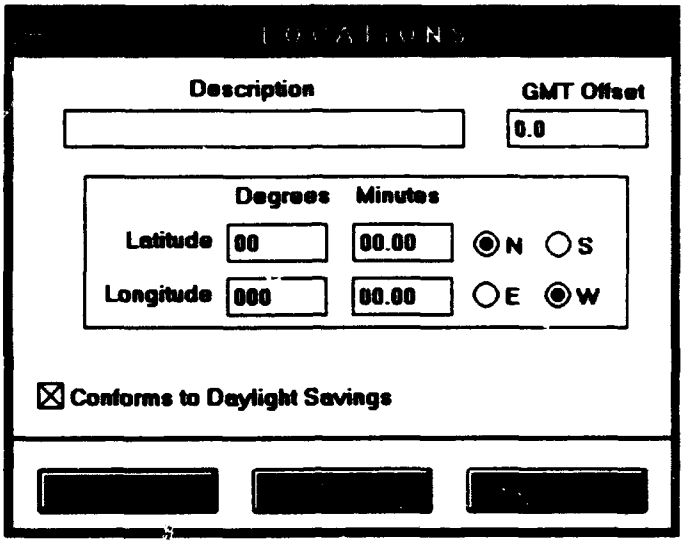
What happens if the location for which we desire data isn't in MOONLITE's database?

That is easy! We just add it!

From the main MOONLITE Screen, choose the Locations menu option. The locations menu allows you to Add, Edit, or Delete a location from MOONLITE's database.

Let's add a location by selecting "Add" from the menu.

MOONLITE presents the Add Location dialog box to allow us to add our location.



Here we need to fill in an english description of the location, how many hours the location is offset from Greenwich Mean Time, the location's latitude and longitude, and whether or not this location uses daylight savings time.

The first item we need to enter is the name of the location we are adding. The DESCRIPTION box should already be highlighted. If it is not, we can move to it by pressing TAB to move to the next box, or SHIFT-TAB to move to the previous box. When the Description box is highlighted, type the location "Test Location". We will use this location just to demonstrate. When you are done typing, press TAB to move to the next field.

The GMT Offset box in MOONLITE allows you to enter a number with a decimal point. This is useful if the location you are adding does not conform to the hourly standard. For example, some places in Norway are offset from Greenwich Time by 45 minutes. If this were the case, you would enter 0.75 for the GMT Offset. For most locations, though, you will enter a whole number. Let's assume that "Test Location" is located in California. California is offset eight hours from Greenwich Time, so we will enter an "8.0" in the GMT Offset box and press TAB to move to the next field.

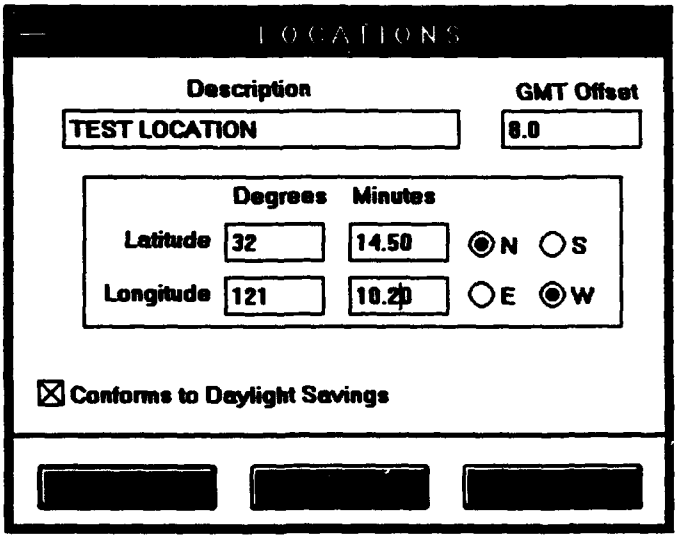
In the Latitude and Longitude boxes values are entered in degrees, minutes and tenths of minutes. If you have a location specified by minutes and seconds, you will have to convert to minutes and

tenths of minutes. This is a simple conversion accomplished by dividing the number of seconds bu 60. The answer is tenths of a minute. For example, assume a location of 43° 20' 30". This should convert to 43° 20.5'.

The last field of the Add Location Dialog Box is one entitled "Conforms to Daylight Savings Time." By checking this box, MOONLITE will add one hour to all input and output times when a corresponding box entitled "Use Daylight Savings Time" is checked on the Spot Data, Position Chart, Planning Calendar, or Daily Events dialog boxes. If this box is not checked, then even if you check the "Use Daylight Savings Time", the time will not be incremented. In this way, you may tag "Use Daylight Savings Time" when Daylight Savings is in effect and know that only those locations that conform to Daylight Savings will be affected.

After you have finished filling out the dialog it will look like the one below. If you are satisfied with the entries, press "Save" to save this data. If any of the numbers you have entered are outside of allowed parameters (for example, a latitude greater than 90 degrees), MOONLITE will warn you that an entry was not allowed and return you to the field that was in error.

Your dialog box may look slightly different depending on what numbers you input for the latitude and longitude of your "Test Location".



We should point out here that the Test Location you just entered is only stored in memory. It has not been saved to the disk yet. If you try to quit MOONLITE or open a new location file, you will receive a warning from MOONLITE that your locations have changed. MOONLITE will give you the opportunity to save your changes or to discard them before continuing.

If you want to save the location you just entered, you can use the menu option FILE|SAVE to save all of the locations currently in memory to the default file moonlite.dat. If you wish to save the locations in a different file, you can use the menu option FILE | SAVE AS. SAVE AS allows you to specify a file name and directory for your data file.

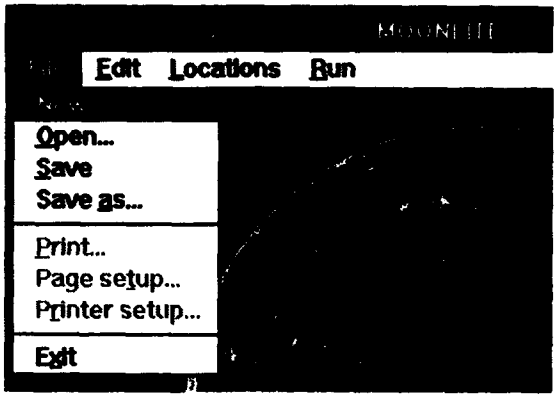
At the time this manual was written, the Position Chart and Planning Calendar are not fully implemented. Descriptions of their operation will be added after completion.

Menu Options

The File Menu

The File menu contains options for opening and saving location data files, seting up printer options, printing, and for closing the MOONLITE program.

The File menu looks looks like this:



File | New

This option is used to create a new locations data file. The new location data file will be empty. See LOCATIONS|ADD to learn how to add locations to a data file.

File | Open

This option is used to open an existing locations data file. When you select this option, you will be presented with a Windows file selection dialog box. When you have selected a data file, MOONLITE will open that file and read the locations into memory.

File | Save

This option is used to save the locations presently in memory to the hard drive or floppy disk. Selecting this option causes MOONLITE to overwrite the file that is currently in use. For example, if MOONLITE is using moonlite.dat for a data file and you have added locations, then selected FILE|SAVE, MOONLITE will overwrite moonlite.dat with the new locations. All locations currently in memory will be written to the file. The effect is that you have just added your new locations to the locations that were previously stored in moonlite.dat. This is the recommended method for adding locations to the data file.

File | Save as...

This option is used to save the locations currently in memory into a file other than the one currently in use. For example, if MOONLITE is using the file moonlite.dat as a data file, and you added some new locations and want them saved to a file named something other than moonlite.dat, you would choose FILE|SAVE AS....

File | Printer setup...

This option allows you to determine the way MOONLITE will interact with your printer. It allows you to select printers, and to access their options menu. Using this menu you will be able to determine print quality and orientation.

File | Exit

This option is used to exit the MOONLITE program. If you have made any changes to the locations in memory, MOONLITE will

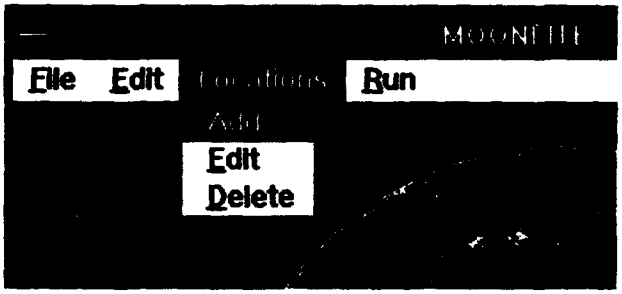
warn you that your changes are not saved and ask whether or not you want MOONLITE to save the files for you. You may also quit MOONLITE by double clicking in the upper-left hand corner of the main MOONLITE window.

The Edit Menu

The only option available on the Edit Menu is Copy. You may use this option to copy information displayed in MOONLITE to the standard Windows clipboard for inclusion in other applications such as Word processors.

The Locations Menu

The Locations Menu Looks like this:



Locations | Add

The LOCATIONS|ADD option is used to add new locations to the list of locations currently stored in memory. MOONLITE will store a (theoretically) unlimited number of locations and is constrained only by the amount of memory available. New locations added via the LOCATIONS|ADD option are not automatically saved to disk. To save additions you must use either the FILE|SAVE or FILE|SAVE AS options.

Locations | Edit

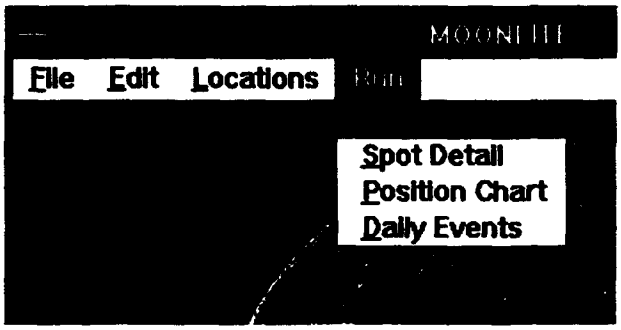
The LOCATIONS|EDIT option is used to Edit locations already in memory. To edit a location stored in a data file on disk, you must open the file for use, edit the desired locations, and save the file back to disk.

Locations | Delete

The LOCATIONS|DELETE option is used to remove locations from memory. To remove a location from a data file, you must open the file for use, delete the desired locations, and save the file back to disk. The LOCATIONS|DELETE dialog box is identical to the LOCATIONS|ADD and LOCATIONS|EDIT dialog boxes with the exception that the Save button has been replaced with a Delete button. You select a location to delete just as you do a location to edit, buy using the pull-down list box labeled "Description".

The Run Menu

The run menu is the heart of MOONLITE. It is here that you will choose the output you desire and initiate the dialog boxes for the different Run options. The Run menu looks like this:



Run | Calendar

The RUN|CALENDAR option is probably the most used option of MOONLITE. You use this option to generate a light level planning calendar.

This option is not yet fully implemented and will be further expounded upon when completed.

Run | Spot Detail

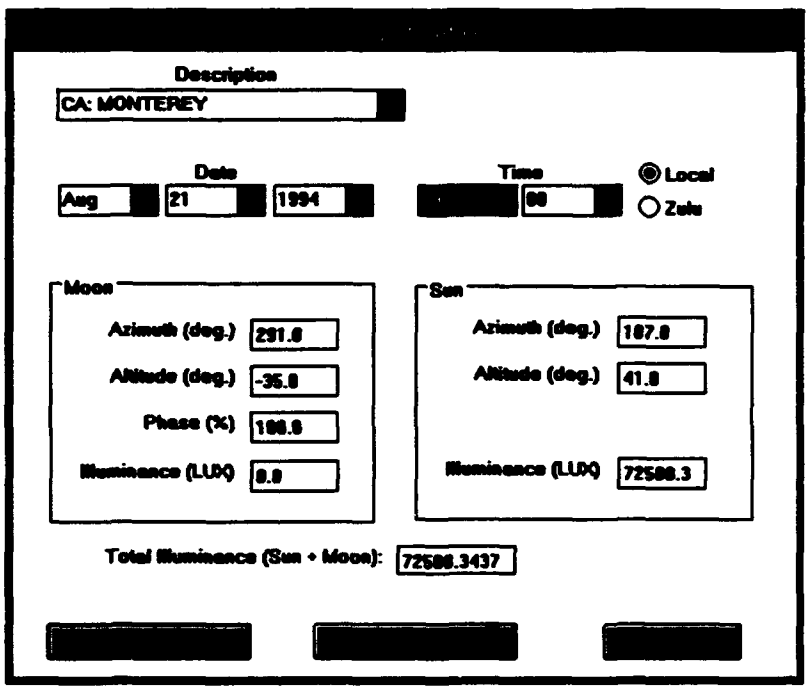
The RUN|SPOT DETAIL option is used to access information about a particular location at a particular time. This option requires you to choose a location, a date, and a time. It returns the

azimuth, altitude and illuminance of the sun and moon, the total illuminance of the sun and moon combined, and the phase of the moon.

You may choose to look at many different locations, dates, or times before closing this dialog. Each time you change any parameter, MOONLITE will recalculate the data on the fly and display the results. When you are finished with the dialog, click on the DONE button.

If you wish a print-out of the data, click on the PRINT button. MOONLITE uses the default printer selected for Windows.

A HELP button is provided should you have any question about any option or parameter in the Spot Detail dialog box.



Run | Position Chart

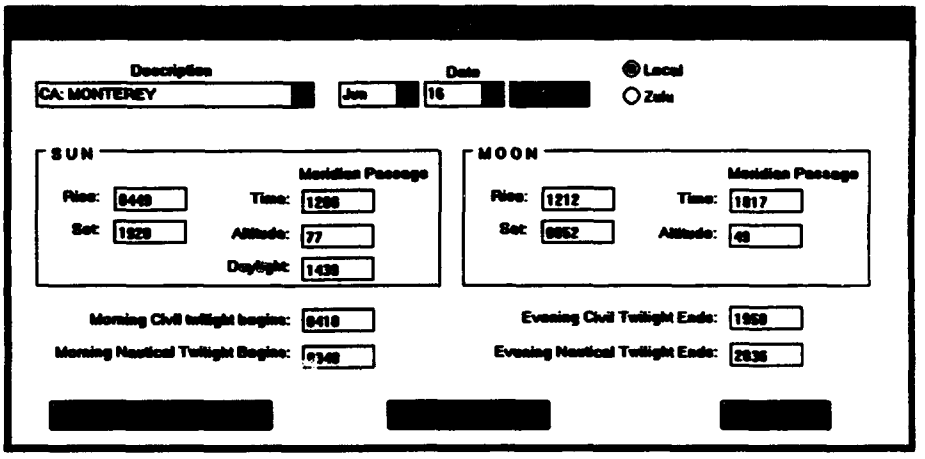
This option is not yet fully implemented and will be further expounded upon when completed.

Run | Daily Events

The RUN|DAILY EVENTS option is used to access information about a particular location. The option requires you to select a location and a date. MOONLITE will respond with the time of Sunrise, Sunset, Moonrise, Moonset, meridian passage of both the sun and moon, and the times of the beginning and end of civil and nautical twilight.

If you wish a print-out of the data, click on the PRINT button. MOONLITE uses the default printer selected for Windows.

A HELP button is provided should you have any question about any option or parameter in the Spot Detail dialog box.



The Help Menu

This option is not yet fully implemented and will be further expounded upon when completed.

APPENDIX B: SOURCE CODE

BITMAP.H

```
//
//    Copyright (c) 1994 Michael t. Lester
//    All rights reserved.
//

#include"zapp.hpp"

class BmpShowPane : public zPane {
        zBitmap *bmp;
        zBitmapDisplay *bmpdisp;
        BOOL fileOpen;
    public:
        BmpShowPane(zWindow*, zSizer*);
        ~BmpShowPane();
        void display(char*);
        int draw(zDrawEvt*);
};
```

CONSTANT.H

```
/*************************************************************************

constants.h

Constants needed moonlite.cpp and associated files

*************************************************************************/

#ifndef constant_h
#define constant_h

char *months[] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun","Jul", "Aug", "Sep", "Oct", "Nov", "Dec", 0};

char *days28[] = {"01", "02", "03", "04", "05", "06", "07", "08", "09", "10",
                  "11", "12", "13", "14", "15", "16", "17", "18", "19", "20",
                  "21", "22", "23", "24", "25", "26", "27", "28", 0};

char *days29[] = {"01", "02", "03", "04", "05", "06", "07", "08", "09", "10",
                  "11", "12", "13", "14", "15", "16", "17", "18", "19", "20",
                  "21", "22", "23", "24", "25", "26", "27", "28", "29", 0};

char *days30[] = {"01", "02", "03", "04", "05", "06", "07", "08", "09", "10",
                  "11", "12", "13", "14", "15", "16", "17", "18", "19", "20",
                  "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", 0};

char *days31[] = {"01", "02", "03", "04", "05", "06", "07", "08", "09", "10",
                  "11", "12", "13", "14", "15", "16", "17", "18", "19", "20",
                  "21", "22", "23", "24", "25", "26", "27", "28", "29", "30",
                  "31", 0};

char *years[] = {  "1986", "1987", "1988", "1989", "1990", "1991", "1992", "1993", "1994", "1995", "1996",
"1997",                                "1998", "1999", "2000", "2001", "2002", "2001", "2002", "2003", "2004",
"2005", 0};

char *hours[] = { "00", "01", "02", "03", "04", "05", "06", "07", "08", "09",
                  "10", "11", "12", "13", "14", "15", "16", "17", "18", "19",
                  "20", "21", "22", "23", "24", 0};

char *minutes[] = {     "00", "01", "02", "03", "04", "05", "06", "07", "08", "09",
                        "10", "11", "12", "13", "14", "15", "16", "17", "18", "19",
                        "20", "21", "22", "23", "24", "25", "26", "27", "28", "29",
                        "30", "31", "32", "33", "34", "35", "36", "37", "38", "39",
                        "40", "41", "42", "43", "44", "45", "46", "47", "48", "49",
                        "50", "51", "52", "53", "54", "55", "56", "57", "58", "59",
                        0};

char *resolution[] = {"1", "5", "10", "15", 0};
```

```
char *GMToffset[] = { "00", "01", "02", "03", "04", "05", "06", "07", "08", "09",
                      "10", "11", "12", 0};

#endif //define constant
```

DEFINES.H

```
/*****************************************************************************
*****

constants.h

Constants needed moonlite.cpp and associated files

*****************************************************************************
*****/

#ifndef defines_h
#define CM_POPUPITEM    101
#define defines_h

#define TRUE                                1
#define FALSE                               0

#define MENU_MAIN                           100
#define ID_MENU_POSITION                    118
#define ID_MENU_FILENEW                     101
#define ID_MENU_FILEOPEN                    102
#define ID_MENU_FILESAVE                    103
#define ID_MENU_FILESAVEAS                  104
#define ID_MENU_FILEPRINT                   105
#define ID_MENU_FILEPAGESETUP               106
#define ID_MENU_FILEPRINTERSETUP  107
#define ID_MENU_FILEEXIT                    108
#define ID_MENU_EDITUNDO                    109
#define ID_MENU_EDITCUT                     110
#define ID_MENU_EDITCOPY                    111
#define ID_MENU_EDITPASTE                   112
#define ID_MENU_LOCS_ADD                    113
#define ID_MENU_LOCS_EDIT                   114
#define ID_MENU_LOCS_DELETE                 115
#define ID_MENU_CALENDAR                    116
#define ID_MENU_SPOTDATA                    117
```

```
#define ID_MENU_DAILYEVENTS             119
#define ID_MENU_HELPINDEX               120
#define ID_MENU_HELPKEYBOARD            121
#define ID_MENU_HELPCOMMANDS            122
#define ID_MENU_HELPPROCEDURES          123
#define ID_MENU_HELPUSINGHELP           124
#define ID_MENU_HELPABOUT               125

#define DIALOG_SETTINGS                 200
#define ID_S_BEGINMONTH                 201
#define ID_S_BEGINDAY                   202
#define ID_S_BEGINYEAR                  203
#define ID_S_BEGINHOUR                  204
#define ID_S_BEGINMIN                   205
#define ID_S_ENDMONTH                   206
#define ID_S_ENDDAY                     207
#define ID_S_ENDYEAR                    208
#define ID_S_ENDHOUR                    209
#define ID_S_ENDMIN                     210
#define ID_S_LOCATION                   211
#define ID_S_THRESHOLD                  212
#define ID_S_PERIOD                     213
#define ID_S_FAST                       214
#define ID_S_ACCURATE                   215
#define ID_S_LOCAL                      216
#define ID_S_GREENWICH                  217
#define ID_S_PRINT2FILE                 218

#define DIALOG_LOCATIONS                300
#define ID_L_DESC                       301
#define ID_L_GMT                        302
#define ID_L_LATDEG                     303
#define ID_L_LATMIN                     304
#define ID_L_N                          305
#define ID_L_S                          306
#define ID_L_LONDEG                     307
#define ID_L_LONMIN                     308
#define ID_L_E                          309
#define ID_L_W                          310
#define ID_L_DST                        311
```

```
#define LOCATION_DELETE         400
#define ID_LD_DESC              401
#define ID_LD_GMT               402
#define ID_LD_LATDEG            403
#define ID_LD_LATMIN            404
#define ID_LD_N                 405
#define ID_LD_S                 406
#define ID_LD_LONDEG            407
#define ID_LD_LONMIN            408
#define ID_LD_E                 409
#define ID_LD_W                 410
#define ID_LD_DST               411

#define EDIT_LOCATION           500
#define ID_ED_DESC              501
#define ID_ED_GMT               502
#define ID_ED_LATDEG            503
#define ID_ED_LATMIN            504
#define ID_ED_N                 505
#define ID_ED_S                 506
#define ID_ED_LONDEG            507
#define ID_ED_LONMIN            508
#define ID_ED_E                 509
#define ID_ED_W                 510
#define ID_ED_DST               511

#define SPOT_DETAIL             600
#define ID_SPOT_DESC            601
#define ID_SPOT_MONTH           602
#define ID_SPOT_DAY             603
#define ID_SPOT_YEAR            604
#define IDC_GROUPBOX3           605
#define IDC_GROUPBOX4           606
#define ID_SPOT_HOUR            605
#define ID_SPOT_MIN             606
#define ID_SPOT_LOCAL           607
#define ID_SPOT_ZULU            608
#define ID_SPOT_MAZ             609
#define ID_SPOT_MAL             610
```

```
#define ID_SPOT_MPR                611
#define ID_SPOT_MIL                612
#define ID_SPOT_SAZ                613
#define ID_SPOT_SAL                614
#define ID_SPOT_SIL                615
#define ID_SPOT_TIL                616
#define ID_SPOT_PRINT              617

#define EVENTS                     700
#define ID_EVENT_DESC              701
#define ID_EVENT_MONTH             702
#define ID_EVENT_DAY               703
#define ID_EVENT_YEAR              704
#define ID_EVENT_SUNRISE           705
#define ID_EVENT_SUNSET            706
#define ID_EVENT_SUNTIME           707
#define ID_EVENT_SUNALT            708
#define ID_EVENT_MOONRISE          709
#define ID_EVENT_MOONSET           710
#define ID_EVENT_MOONTIME          711
#define ID_EVENT_MOONALT           712
#define ID_EVENT_AMCIV             713
#define ID_EVENT_AMNAT             714
#define ID_EVENT_PMCIV             715
#define ID_EVENT_PMNAT             716
#define ID_EVENT_PRINT             717
#define IDC_GROUPBOX1              718
#define IDC_GROUPBOX2              719
#define ID_EVENT_DAYLIGHT          720
#define ID_EVENT_LOCAL             721
#define ID_EVENT_ZULU              722

#define POSITION                   800
#define ID_POS_DESC                801
#define ID_POS_MONTH               802
#define ID_POS_DAY                 803
#define ID_POS_YEAR                804
#define ID_POS_BHOUR               805
#define ID_POS_BMIN                806
#define ID_POS_DONE                812
```

```
#define ID_POS_LOCAL                    809
#define ID_POS_ZULU                     810
#define ID_POS_RES                      811
#define ID_POS_DONE                     812

#define Position_Chart                  900
#define POS_CHART_LATLON                901
#define POS_CHART_DATE                  902
#define POS_CHART_ARRAY                 903
#define POS_CHART_PRINT                 904
#define POS_CHART_DONE                  905

#define ICON_1                           9999

#define DESC_LENGTH                     30


#endif //define constant
```

DIALOGS.H

```
//
// dialogs.h
//

#ifndef dialogs.h
#define dialogs.h

#include "ll.h"
#include "zapp.hpp"
#include "defines.h"
#include "ll.h"


class C_dlg_locations : public zFormDialog {
      friend class locations_list;
      locations_list *ll;
      zString _location;
      float _latmin, _lonmin, _GMT;
      int _E, _N, _DST,_latdeg, _londeg;
      location_struct data;
      zDefPushButton *Ok_Pressed;

public:
      C_dlg_locations(locations_list *,zWindow *,zResId&);
      int doOK();
      char *location() { return(char *)_location;};
      int find(char *);
};

class C_dlg_loc_del : public zFormDialog {

      friend class locations_list;
      locations_list *ll;
      zString _location;
      float _latmin, _lonmin, _GMT;
      int _E, _N, _DST, _latdeg, _londeg;
      location_struct data;
      node *temp;
      zDefPushButton *Ok_Pressed;
      zComboBox *lb_desc;
      zIntEdit *Latdeg, *Londeg;
      zFloatEdit *Latmin, *Lonmin, *UGMT;
      zCheckBox *UDST;
      zRadioGroup *UEW, *UNS;

 public:
      C_dlg_loc_del(locations_list *, zWindow *,zResId&, node*);
      char *location() { return(char *)_location;};
      lbChanged(zEvent *);
```

```
};

class C_dlg_loc_edit : public zFormDialog {

    friend class locations_list;
    locations_list *ll;
    zString _location;
    float _latmin, _lonmin, _GMT;
    int _E, _N, _DST, _latdeg, _londeg;
    location_struct data;
    node *temp;
    zDefPushButton *Ok_Pressed;
    zComboBox *lb_desc;
    zIntEdit *Latdeg, *Londeg;
    zFloatEdit *Latmin, *Lonmin, *UGMT;
    zCheckBox *UDST;
    zRadioGroup *UEW, *UNS;

public:
    C_dlg_loc_edit(locations_list *, zWindow *,zResId&, node*);
    char *location() { return(char *)_location;};
    lbChanged(zEvent *);
};

class C_dlg_spotdata : public zFormDialog {

    locations_list *ll;
    zString _location, temp_desc;
    float _latmin, _lonmin, _GMT;
    int _E, _N, _DST, _latdeg, _londeg;
    float _saz, _sal, _sil, _maz, _mal, _mil, _mpr, _til;
    int _month, _day, _year, _hour, _min, _local;
    float lo, f, iy, z, h;
    int id, im, test;
    location_struct data;
    zDefPushButton *Ok_Pressed;
    zComboBox *lb_desc, *lb_month, *lb_day, *lb_year, *lb_hour, *lb_min;
    zFloatEdit *saz, *sal, *sil, *maz, *mal, *mil, *mpr, *til;
    zRadioGroup *local;

public:
    C_dlg_spotdata(locations_list *,zResId&);
    //char *location() { return(char *)_location;};
    lbChanged(zEvent *);
};

class C_dlg_Event : public zFormDialog {

    locations_list *ll;
    zString _location, temp_desc;
    float _latmin, _lonmin, _GMT;
    int _E, _N, _DST, _latdeg, _londeg;
    float _sr, _sst, _st, _sa, _mr, _ms, _mt, _ma, _ac, _an, _pc, _pn, _dl;
```

```
    int _month, _day, _year, _hour, _min, _local;
    float lo, f, iy, z, h;
    int id, im, test;
    location_struct data;
    zDefPushButton *Ok_Pressed;
    zComboBox *lb_desc, *lb_month, *lb_day, *lb_year, *lb_hour, *lb_min;
    zFloatEdit *sr, *sst, *st, *sa, *mr, *ms, *mt, *ma, *ac, *an, *pc, *pn, *dl;
    zRadioGroup *local;

public:
    C_dlg_Event(locations_list *,zResId&);
    //char *location() { return(char *)_location;};
    lbChanged(zEvent *);
};

class C_dlg_Pos : public zFormDialog{

    zPushButton *Done;
    zComboBox *lb_desc, *lb_month, *lb_day, *lb_year, *lb_res,
              *lb_bhour, *lb_bmin, *lb_ehour, *lb_emin;
    zRadioGroup *local;
    locations_list *ll;
    zString _location, temp_desc;
    float _latmin, _lonmin, _GMT;
    int _E, _N, _DST, _latdeg, _londeg;
    float _saz, _sal, _sil, _maz, _mal, _mil, _mpr, _til;
    int _month, _day, _year, _bhour, _bmin, _ehour, _emin, _local, _res;
    float lo, f, iy, z, h, delta;
    int id, im, test;
    zString monthstr;
    zString daystr;
    zString yearstr;
    zTextPane *tp;
    zString temp, trash;
    int count;
    location_struct data;

    //print_desc(zTextPane *);

    public:

    C_dlg_Pos(locations_list *, zResId&);
    int doDone(zEvent *);
    lbChanged(zEvent *);
};


/*class POS_Show : public Pos_Base {

    zPushButton *Ok_Pressed;
    zComboBox *array;
```

```
public:
    POS_Show(zResId&);
    zComboBox *display_array;
};
*/


#endif //dialogs.h
```

FASTAL.H

```
/*
 *   fastcal.h
 *
 *   Description:   Implimentation of Algorithms of Circular 171
 *
 *   Input:         As shown for each member function.
 *
 *   Output:        reference variables used for ease of data transfer
 *
 *   Comments:
 *
 *   Michael T. Lester              05 Nov 1993
 *
 *        Modified                                        21 Feb 1994
 *
 */

#ifndef fastal.h
#define fastal.h

class fast_algorithm
{

    private:

        double RD, DR, CE, SE, A[4], B[2], LI, SI, CI, J, Z0, TD, T, AS1, Y;
        double E, D, G, LS, SD, DS, V, Q, W, SB, CB, X, O, S, CD, SV, P;
        double CS, AZ, HA, U, M, IS1, IL, DT, R, C;
        int   K;

    public:

        void illum( float   LO,     //Longitude, degrees, E=+, W=-
                    float   F,              //Latitude, degrees, N=+, E=-
                    float   IY,     //Year (####)
                    int             IM,     //Month (01 - 12)
                    int             ID,     //Day (01 - 31) assumed correct for calendar
                    float   Z,              //Time format, 0=Zulu, 1=local (zone)
                    float   H,              //Hour, 24 hr format, zulu or local as above
                    float&  saz,  //result, Sun Azimuth
                    float&  sal,    //result, Sun Altitude
                    float&  sil,    //result, Sun Illuminance
                    float&  maz,    //result, Moon Azimuth
                    float&  mal,    //result, Moon Altitude
                    float&  mil,    //result, Moon Illuminance
                    float&  mpr,    //result, Moon Percentage of Full
                    float&  til     //result, Total Illuminance
                    );
```

```
        void event(  float       LO,    //Longitude, degrees, E=+, W=-
                     float       F,             //Latitude, degrees, N=+, E=-
                     float       IY,    //Year (####)
                     int         IM,    //Month (01 - 12)
                     int         ID,    //Day (01 - 31) assumed correct for calendar
                     float       Z,             //Time format, 0=Zulu, 1=local (zone)
                     float       H,     //Hours from zulu
                     float&      sr,    //result, Sun Rise
                     float&      sst,   //result, Sun Set Time
                     float&      st,    //result, Time of Sun Meridian Passage
                     float&      sa,    //result, Altitude of Sun Meridian Passage
                     float&      mr,    //result, Moon Rise
                     float&      ms,    //resuti, Moon Set
                     float&      mt,    //result, Time of Moon Meridian Passage
                     float&      ma,    //result, Altitude of Moon Meridian Passage
                     float&      ac,    //result, Morning Civil Twilight
                     float&      an,    //result, Morning Nautical Twilight
                     float&      pc,    //result, Evening Civil Twilight
                     float&      pn,    //result, Evening Nautical Twilight
                     float&      dl             //result, length of daylight
                     );
};

#endif //#define fastai.h
```

LL.H

```
//
// ll.h
//
// header file for ll.cpp, the linked list routines for storing the location data.
//
#ifndef ll_h
#define ll_h

#include "zapp.hpp"
#include "defines.h"

 struct location_struct
     {
     char      desc[DESC_LENGTH + 1];
     float latdeg;
     float latmin;
     int   NS;
     float londeg;
     float lonmin;
     int   EW;
     float GMT;
     int   DST;
     };

 class node
{
     public:
          node();
          ~node();
          location_struct *data;
          node *next;
};

     /* Class locations handles all aspects of adding, editing, saving and
     returning information from the locations file.
     */

class locations_list
{
     private:
          zString   datafile;
          node           *head, *temp_ptr;
          int       list_dirty;

     public:
          locations_list();
          //~locations_list();
          int clear();
          int save();
          int savess(zWindow *pwin);
```

```
        int open_dialog(zWindow *pwin);
        int open();
        int edit_dialog(zWindow *pwin, locations_list *);
        int add_dialog(zWindow *pwin, locations_list *);
        int add_record(location_struct *);
        int delete_dialog(zWindow *pwin, locations_list *);
        node* find(zString &);
        int peek_list_dirty() {return list_dirty;};
        void reset_pointer(){temp_ptr = head;};
        char* get_next();
};


#endif //define ll.h
```

LOCDATA.H

```
//
// locdata.h
//
#ifndef locdata.h
#define locdata.h

location_struct locdata[] =
    {
    {"CA: MCAS Tustin", 33, 42.2, 1, 117, 49.6, 1, 8, 1, 1},
    {"CA: NAS Noth Island", 32, 41.9, 1, 117, 12.8, 1, 8, 1, 1},
    {"CA: NAS Miramar", 32, 52.1, 1, 117, 08.5, 1, 8, 1, 1},
    {"CA: ALF Hunter Ligett (TUSI)", 36, 0.0, 1, 121, 14.0, 1, 8, 1, 1}
    };

#endif //define locdata.h
```

MENUFRAM.H

```
//
// menuframe.h
//
#ifndef menufram.h
#define menufram.h

#include "zapp.h"

class MenuFrame : public zAppFrame{
    zTextPane* tp;
public:
    MenuFrame(zWindow* parent,zSizer* siz,DWORD winStyle,const char* title);
    ~MenuFrame();
    virtual int command(zCommandEvt *);
    int doExit(zCommandEvt *);
    int AddLocation(zCommandEvt *);
    doOk();
};

#endif //menufram
```

ROUTINES.H

```
//
// routines.h
//

#ifndef routines_h
#define routines_h

#include "ll.h"

class routines {

    private:

        locations_list *ll;

    public:
        routines(locations_list *list_in);
        int spotdata();
        int event();
        int position();
    };


#endif //routines_h
```

BMPSHOW.CPP

```
//
//    Copyright (c) 1992, 1993 Inmark Development Corp.
//    All rights reserved.
//

#include "zapp.hpp"
#include "bmpshow.h"

BmpShowPane::BmpShowPane(zWindow *w, zSizer *s):zPane(w,s) {
     bmp = 0;
     bmpdisp = 0;
     fileOpen = FALSE;
}

BmpShowPane::~BmpShowPane() {
     if (bmpdisp) delete bmpdisp;
}

void BmpShowPane::display(char *name) {
     if (bmpdisp) delete bmpdisp;
     bmp = new zBitmap(canvas(), name);
     bmpdisp = new zBitmapDisplay(bmp);
     fileOpen = TRUE;
     canvas()->setDirty();
}

int BmpShowPane::draw(zDrawEvt*) {
     if (fileOpen) {
          zDimension dm = bmp->size();
          canvas()->lock();
          bmpdisp->copyTo(canvas(),0,0,dm.width(),dm.height(),0,0);
          canvas()->unlock();
     }
     return 1;
}
```

DIALOGS.CPP

```
//
// dialogs.cpp
//
// routines for dialogs.cpp
//

#include <string.h>
#include <math.h>
#include <stdlib.h>
#include "constant.h"
#include "dialogs.h"
#include "ll.h"
#include "fastal.h"


C_dlg_locations::C_dlg_locations(locations_list *all, zWindow *w,zResId& rid)
                                : zFormDialog(w,rid) {
    ll = all;
    _location = "";
    _GMT = 0.0;
    _latdeg = 0;
    _latmin = 0.0;
    _E = ID_L_W;
    _londeg = 0;
    _lonmin = 0.0;
    _N = ID_L_N;
    _DST = 1;

    Ok_Pressed = new zDefPushButton(this, IDOK);
    Ok_Pressed->setNotifyClicked(this, (ClickProc)&C_dlg_locations::doOK);

    new zStringEdit(this, ID_L_DESC, &_location, "!(30)");
    new zIntEdit(this, ID_L_LATDEG, &_latdeg, "00", FLD_NOTREQUIRED);
    new zFloatEdit(this, ID_L_LATMIN , &_latmin, "00.00", FLD_NOTREQUIRED);
    new zIntEdit(this, ID_L_LONDEG, &_londeg, "000", FLD_NOTREQUIRED);
    new zFloatEdit(this, ID_L_LONMIN, &_lonmin, "00.00", FLD_NOTREQUIRED);
    new zCheckBox(this,ID_L_DST, &_DST);
    new zRadioGroup(this, ID_L_E, ID_L_W, &_E);
    new zRadioGroup(this, ID_L_N, ID_L_S, &_N);
    new zFloatEdit(this, ID_L_GMT , &_GMT, "0.0", FLD_NOTREQUIRED);

    show();
    modal();
}

int C_dlg_locations::doOK()
{
    if(!zFormDialog::storeData())
            {
            zMessage mess(app->rootWindow(),"A Description is Required","",MB_OK);
```

```
            setFocus();
            }
    else if ((_latdeg < 0 ) || (_latdeg > 90 ) )
            {
            zMessage mess(app->rootWindow(),"Degrees Latitude must be between 0 & 90","",MB_OK);
            setFocus();
            }
    else if ((_londeg < 0 ) || (_londeg > 179 ) )
            {
            zMessage mess(app->rootWindow(),"Degrees Longitude must be between 0 &
179","",MB_OK);
            setFocus();
            }
    else if ((_latmin < 0 ) || (_latmin > 59 ) || (_lonmin < 0 ) || (_lonmin > 59 ))
            {
            zMessage mess(app->rootWindow(),"Minutes of Latitude/Longitude must be between 0 &
59","",MB_OK);
            setFocus();
            }
    else if ((_GMT < 0.0 ) || (_GMT > 12.0 ))
            {
            zMessage mess(app->rootWindow(),"GMT Offset must be between 0 and 12","",MB_OK);
            setFocus();
            }
    else if (ll->find(_location))
            {
            zMessage mess(app->rootWindow(),"This Location is already defined.\nPlease select another
name",
                "Duplicate Name!",MB_OK);
            setFocus();
            }
    else
        {
        shutdown();
        return(1);
        }
}


C_dlg_loc_del::C_dlg_loc_del(locations_list *all, zWindow *w,zResId& rid, node *temp) : zFormDialog(w,rid)
{
    ll = all;
    _location = "  [Choose Location To Delete]";
    _GMT = 0.0;
    _latdeg = 0;
    _latmin = 0.0;
    _E = ID_LD_W;
    _londeg = 0;
    _lonmin = 0.0;
    _N = ID_LD_N;
    _DST = 0;

    Latdeg = new zIntEdit(this, ID_LD_LATDEG, &_latdeg);
```

```
    Latmin = new zFloatEdit(this, ID_LD_LATMIN , &_latmin, "###.###", FLD_NOTREQUIRED);
    Londeg = new zIntEdit(this, ID_LD_LONDEG, &_londeg);
    Lonmin = new zFloatEdit(this, ID_LD_LONMIN, &_lonmin, "###.###", FLD_NOTREQUIRED);
    UDST   = new zCheckBox(this,ID_LD_DST, &_DST);
    UEW    = new zRadioGroup(this, ID_LD_E, ID_LD_W, &_E);
    UNS    = new zRadioGroup(this, ID_LD_N, ID_LD_S, &_N);
    UGMT   = new zFloatEdit(this,ID_LD_GMT, &_GMT, "0.0", FLD_NOTREQUIRED);
    lb_desc = new zComboBox(this,ID_LD_DESC, &_location);
    while (temp != NULL)
        {
        lb_desc->add(temp->data->desc);
        temp = temp->next;
        }
    lb_desc->add("  [Choose Location To Delete]");
    lb_desc->setToDefault();
    lb_desc->setNotifySelChange(this, (NotifyProc)&C_dlg_loc_del::lbChanged);

    show();
    modal();
}

int C_dlg_loc_del::lbChanged(zEvent *ce)
    {
        // I have no idea why I have to reset the zString in this place
        // but if I don't, it truncates the value to 15 after the
        // first iteration.
    _location = "                    ";
    lb_desc->getEditText(_location);
    node *desired = ll->find(_location);
    if (desired)
        {
        _latmin = desired->data->latmin;
        _latdeg = desired->data->latdeg;
        _lonmin = desired->data->lonmin;
        _londeg = desired->data->londeg;
        _GMT    = desired->data->GMT;
        _DST    = desired->data->DST;
        _E      = desired->data->EW + 409;
        _N      = desired->data->NS + 405;
        Latdeg->setToDefault();
        Londeg->setToDefault();
        UDST->setToDefault();
        UGMT->setToDefault();
        UEW->setToDefault();
        UNS->setToDefault();
        Lonmin->setToDefault();
        Latmin->setToDefault();
        }
    }

C_dlg_loc_edit::C_dlg_loc_edit(locations_list *all, zWindow *w,zResId& rid, node *temp) :
zFormDialog(w,rid) {
    ll = all;
```

```
    _location = "  [Choose Location To Edit]";
    _GMT = 0.0;
    _latdeg = 0;
    _latmin = 0.0;
    _E = ID_ED_W;
    _londeg = 0;
    _lonmin = 0.0;
    _N = ID_ED_N;
    _DST = 0;

    Latdeg = new zIntEdit(this, ID_ED_LATDEG, &_latdeg);
    Latmin = new zFloatEdit(this, ID_ED_LATMIN , &_latmin, "###.###", FLD_NOTREQUIRED);
    Londeg = new zIntEdit(this, ID_ED_LONDEG, &_londeg);
    Lonmin = new zFloatEdit(this, ID_ED_LONMIN, &_lonmin, "###.###", FLD_NOTREQUIRED);
    UDST   = new zCheckBox(this,ID_ED_DST, &_DST);
    UEW    = new zRadioGroup(this, ID_ED_E, ID_ED_W, &_E);
    UNS    = new zRadioGroup(this, ID_ED_N, ID_ED_S, &_N);
    UGMT   = new zFloatEdit(this,ID_ED_GMT, &_GMT, "0.0", FLD_NOTREQUIRED);
    lb_desc = new zComboBox(this,ID_ED_DESC, &_location);
    while (temp != NULL)
        {
        lb_desc->add(temp->data->desc);
        temp = temp->next;
        }
    lb_desc->add("  [Choose Location To Edit]");
    lb_desc->setToDefault();
    lb_desc->setNotifySelChange(this, (NotifyProc)&C_dlg_loc_edit::lbChanged);

    show();
    modal();
}

int C_dlg_loc_edit::lbChanged(zEvent *ce)
    {
        // I have no idea why I have to reset the zString in this place
        // but if I don't, it truncates the value to 15 after the
        // first iteration.
    _location = "                    ";
    lb_desc->getEditText(_location);
    node *desired = ll->find(_location);
    if (desired)
        {
        _latmin = desired->data->latmin;
        _latdeg = desired->data->latdeg;
        _lonmin = desired->data->lonmin;
        _londeg = desired->data->londeg;
        _GMT    = desired->data->GMT;
        _DST    = desired->data->DST;
        _E      = desired->data->EW + 509;
        _N      = desired->data->NS + 505;
        Latdeg->setToDefault();
        Londeg->setToDefault();
        UDST->setToDefault();
```

```
        UGMT->setToDefault();
        UEW->setToDefault();
        UNS->setToDefault();
        Lonmin->setToDefault();
        Latmin->setToDefault();
        }
    }

C_dlg_spotdata::C_dlg_spotdata(locations_list *all, zResId& rid)
                    : zFormDialog(app->rootWindow(),rid) {
    ll = all;
    _location = "    [ Choose Location ]";
    _GMT = 0.0;
    _latdeg = 0;
    _latmin = 0.0;
    _E = ID_LD_W;
    _londeg = 0;
    _lonmin = 0.0;
    _N = ID_LD_N;
    _DST = 0;
    _local = ID_SPOT_LOCAL;
    _day = 0;

    saz = new zFloatEdit(this, ID_SPOT_SAZ , &_saz, "###.###", FLD_NOTREQUIRED);
    sal = new zFloatEdit(this, ID_SPOT_SAL , &_sal, "-###.###", FLD_NOTREQUIRED);
    sil = new zFloatEdit(this, ID_SPOT_SIL , &_sil, "########.###", FLD_NOTREQUIRED);
    maz = new zFloatEdit(this, ID_SPOT_MAZ , &_maz, "###.###", FLD_NOTREQUIRED);
    mal = new zFloatEdit(this, ID_SPOT_MAL , &_mal, "-###.###", FLD_NOTREQUIRED);
    mil = new zFloatEdit(this, ID_SPOT_MIL , &_mil, "########.###", FLD_NOTREQUIRED);
    mpr = new zFloatEdit(this, ID_SPOT_MPR , &_mpr, "###.###", FLD_NOTREQUIRED);
    til = new zFloatEdit(this, ID_SPOT_TIL , &_til, "########.######", FLD_NOTREQUIRED);

    lb_desc = new zComboBox(this, ID_SPOT_DESC, &_location);
    ll->reset_pointer();
    temp_desc = ll->get_next();
    while (temp_desc != "")
        {
        lb_desc->add(temp_desc);
        temp_desc = ll->get_next();
        }
    lb_desc->add("    [ Choose Location ]");
    lb_desc->setToDefault();
    lb_desc->setNotifySelChange(this, (NotifyProc)&C_dlg_spotdata::lbChanged);

    lb_month = new zComboBox(this, ID_SPOT_MONTH, &_month);
    lb_month->addCharStrings(months);
    lb_month->setToDefault();
    lb_month->setNotifySelChange(this, (NotifyProc)&C_dlg_spotdata::lbChanged);

    lb_day = new zComboBox(this, ID_SPOT_DAY, &_day);
    lb_day->addCharStrings(days31);
    lb_day->setToDefault();
    lb_day->setNotifySelChange(this, (NotifyProc)&C_dlg_spotdata::lbChanged);
```

```
    lb_year = new zComboBox(this, ID_SPOT_YEAR, &_year);
    lb_year->addCharStrings(years);
    lb_year->setToDefault();
    lb_year->setNotifySelChange(this, (NotifyProc)&C_dlg_spotdata::lbChanged);

    lb_hour = new zComboBox(this, ID_SPOT_HOUR, &_hour);
    lb_hour->addCharStrings(hours);
    lb_hour->setToDefault();
    lb_hour->setNotifySelChange(this, (NotifyProc)&C_dlg_spotdata::lbChanged);

    lb_min = new zComboBox(this, ID_SPOT_MIN, &_min);
    lb_min->addCharStrings(minutes);
    lb_min->setToDefault();
    lb_min->setNotifySelChange(this, (NotifyProc)&C_dlg_spotdata::lbChanged);

    local = new zRadioGroup(this, ID_SPOT_LOCAL, ID_SPOT_ZULU, &_local);
    local->setNotifyClicked(this, (NotifyProc)&C_dlg_spotdata::lbChanged);

    show();
    modal();
}

int C_dlg_spotdata::lbChanged(zEvent *ce)
    {
    _month = lb_month->selection() + 1;
    _day   = lb_day->selection();
    _year  = lb_year->selection() + 1993;
    _hour  = lb_hour->selection();
    _min   = lb_min->selection();
    switch (_month)
        {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
            {
            lb_day->reset();
            lb_day->addCharStrings(days31);
            lb_day->setToDefault();
            break;
            };
        case 4:
        case 6:
        case 9:
        case 11:
            {
            lb_day->reset();
            lb_day->addCharStrings(days30);
            lb_day->setToDefault();
```

```
                break;
                }
            case 2:
                {
                lb_day->reset();
                if ( float(_year / 4) == (_year / 4.0) &&
                                                                    float(_year / 400)
!= (_year / 400.0) )
                    {
                    lb_day->addCharStrings(days29);
                    }
                else
                    {
                    lb_day->addCharStrings(days28);
                    }
                lb_day->setToDefault();
                }
            }
        storeData();
        if (completed())
            {
            _location = "                    ";
            lb_desc->getEditText(_location);
            node *desired = ll->find(_location);
            if (desired)
                {
                _latmin = desired->data->latmin;
                _latdeg = desired->data->latdeg;
                _lonmin = desired->data->lonmin;
                _londeg = desired->data->londeg;
                _GMT    = desired->data->GMT;
                _DST    = desired->data->DST;
                _E      = desired->data->EW;
                _N      = desired->data->NS;
                }
            lo = _londeg + _lonmin / 60.0;
            if (_E) lo *= -1.0;
            f = _latdeg + _latmin / 60;
            if (_N) f *= -1.0;
            if (_local == 607) z = TRUE;
            else z = FALSE;
            fast_algorithm *alg = new fast_algorithm;
            h = _hour * 100 + _min;
            alg->illum(lo, f, _year+1966, _month+1, _day+1, z, h, _saz, _sal, _sil, _maz, _mal, _mil, _mpr, _til);
            saz->setToDefault();
            sal->setToDefault();
            sil->setToDefault();
            maz->setToDefault();
            mal->setToDefault();
            mil->setToDefault();
            mpr->setToDefault();
            til->setToDefault();
            delete alg;
```

```
        }
    }

C_dlg_Event::C_dlg_Event(locations_list *all, zResId& rid)
                        : zFormDialog(app->rootWindow(),rid) {
    ll = all;
    _location = "    [ Choose Location ]";
    _GMT = 0.0;
    _latdeg = 0;
    _latmin = 0.0;
    _E = ID_LD_W;
    _londeg = 0;
    _lonmin = 0.0;
    _N = ID_LD_N;
    _DST = 0;
    _local = ID_EVENT_LOCAL;
    _day = 0;
    h = 0;

    sr = new zFloatEdit(this, ID_EVENT_SUNRISE,  &_sr, "0000", FLD_NOTREQUIRED);
    sst = new zFloatEdit(this, ID_EVENT_SUNSET,  &_sst, "0000", FLD_NOTREQUIRED);
    st = new zFloatEdit(this, ID_EVENT_SUNTIME,  &_st, "0000", FLD_NOTREQUIRED);
    sa = new zFloatEdit(this, ID_EVENT_SUNALT,   &_sa, "-####", FLD_NOTREQUIRED);
    mr = new zFloatEdit(this, ID_EVENT_MOONRISE, &_mr, "0000", FLD_NOTREQUIRED);
    ms = new zFloatEdit(this, ID_EVENT_MOONSET,          &_ms, "0000", FLD_NOTREQUIRED);
    mt = new zFloatEdit(this, ID_EVENT_MOONTIME, &_mt, "0000", FLD_NOTREQUIRED);
    ma = new zFloatEdit(this, ID_EVENT_MOONALT,          &_ma, "-####", FLD_NOTREQUIRED);
    ac = new zFloatEdit(this, ID_EVENT_AMCIV,    &_ac, "0000", FLD_NOTREQUIRED);
    an = new zFloatEdit(this, ID_EVENT_AMNAT,    &_an, "0000", FLD_NOTREQUIRED);
    pc = new zFloatEdit(this, ID_EVENT_PMCIV,    &_pc, "0000", FLD_NOTREQUIRED);
    pn = new zFloatEdit(this, ID_EVENT_PMNAT,    &_pn, "0000", FLD_NOTREQUIRED);
    dl = new zFloatEdit(this, ID_EVENT_DAYLIGHT, &_dl, "0000", FLD_NOTREQUIRED);

    lb_desc = new zComboBox(this, ID_EVENT_DESC, &_location);
    ll->reset_pointer();
    temp_desc = ll->get_next();
    while (temp_desc != "")
        {
        lb_desc->add(temp_desc);
        temp_desc = ll->get_next();
        }
    lb_desc->add("    [ Choose Location ]");
    lb_desc->setToDefault();
    lb_desc->setNotifySelChange(this, (NotifyProc)&C_dlg_Event::lbChanged);

    lb_month = new zComboBox(this, ID_EVENT_MONTH, &_month);
    lb_month->addCharStrings(months);
    lb_month->setToDefault();
    lb_month->setNotifySelChange(this, (NotifyProc)&C_dlg_Event::lbChanged);

    lb_day = new zComboBox(this, ID_EVENT_DAY, &_day);
    lb_day->addCharStrings(days31);
    lb_day->setToDefault();
```

```
    lb_day->setNotifySelChange(this, (NotifyProc)&C_dlg_Event::lbChanged);

    lb_year = new zComboBox(this, ID_EVENT_YEAR, &_year);
    lb_year->addCharStrings(years);
    lb_year->setToDefault();
    lb_year->setNotifySelChange(this, (NotifyProc)&C_dlg_Event::lbChanged);

    local = new zRadioGroup(this, ID_EVENT_LOCAL, ID_EVENT_ZULU, &_local);
    local->setNotifyClicked(this, (NotifyProc)&C_dlg_Event::lbChanged);

    show();
    modal();
}

int C_dlg_Event::lbChanged(zEvent *ce)
    {
    _month = lb_month->selection() + 1;
    _day   = lb_day->selection();
    _year  = lb_year->selection() + 1993;
    switch (_month)
        {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
            {
            lb_day->reset();
            lb_day->addCharStrings(days31);
            lb_day->setToDefault();
            break;
            };
        case 4:
        case 6:
        case 9:
        case 11:
            {
            lb_day->reset();
            lb_day->addCharStrings(days30);
            lb_day->setToDefault();
            break;
            }
        case 2:
            {
            lb_day->reset();
            if ( float(_year / 4) == (_year / 4.0) &&
                                                                    float(_year / 400)
 != (_year / 400.0) )
                {
                lb_day->addCharStrings(days29);
                }
```

```
                else
                    {
                    lb_day->addCharStrings(days28);
                    }
                lb_day->setToDefault();
                }
            }
        storeData();
        if (completed())
            {
            _location = "                    ";
            lb_desc->getEditText(_location);
            node *desired = ll->find(_location);
            if (desired)
                {
                _latmin = desired->data->latmin;
                _latdeg = desired->data->latdeg;
                _lonmin = desired->data->lonmin;
                _londeg = desired->data->londeg;
                _GMT    = desired->data->GMT;
                _DST    = desired->data->DST;
                _E      = desired->data->EW;
                _N      = desired->data->NS;
                }
            lo = _londeg + _lonmin / 60.0;
            if (_E) lo *= -1.0;
            f = _latdeg + _latmin / 60;
            if (_N) f *= -1.0;
            if (_local == 721) z = TRUE;
            else z = FALSE;
            fast_algorithm *alg = new fast_algorithm;
            alg->event(lo, f, _year+1986, _month+1, _day+1, z, h, _sr, _sst, _st, _sa, _mr, _ms, _mt, _ma,
_ac, _an, _pc, _pn, _dl);
            sr->setToDefault();
            sst->setToDefault();
            st->setToDefault();
            sa->setToDefault();
            mr->setToDefault();
            ms->setToDefault();
            mt->setToDefault();
            ma->setToDefault();
            ac->setToDefault();
            an->setToDefault();
            pc->setToDefault();
            pn->setToDefault();
            dl->setToDefault();
            delete alg;
            }
        }

//----
C_dlg_Pos::C_dlg_Pos(locations_list *all, zResId& rid)
                    : zFormDialog(app->rootWindow(),rid) {
```

```
ll = all;
_location = "    [ Choose Location ]";
_GMT = 0.0;
_latdeg = 0;
_latmin = 0.0;
_E = ID_LD_W;
_londeg = 0;
_lonmin = 0.0;
_N = ID_LD_N;
_DST = 0;
_local = ID_POS_LOCAL;
_day = 0;
_res = 2;
h = 0;

Done = new zPushButton(this, ID_POS_DONE);
Done->setNotifyClicked(this, (ClickProc)&C_dlg_Pos::doDone);

lb_desc = new zComboBox(this, ID_POS_DESC, &_location);
ll->reset_pointer();
temp_desc = ll->get_next();
while (temp_desc != "")
     {
     lb_desc->add(temp_desc);
     temp_desc = ll->get_next();
     }
lb_desc->add("    [ Choose Location ]");
lb_desc->setToDefault();
lb_desc->setNotifySelChange(this, (NotifyProc)&C_dlg_Pos::lbChanged);

lb_month = new zComboBox(this, ID_POS_MONTH, &_month);
lb_month->addCharStrings(months);
lb_month->setToDefault();
lb_month->setNotifySelChange(this, (NotifyProc)&C_dlg_Pos::lbChanged);

lb_day = new zComboBox(this, ID_POS_DAY, &_day);
lb_day->addCharStrings(days31);
lb_day->setToDefault();
lb_day->setNotifySelChange(this, (NotifyProc)&C_dlg_Pos::lbChanged);

lb_year = new zComboBox(this, ID_POS_YEAR, &_year);
lb_year->addCharStrings(years);
lb_year->setToDefault();
lb_year->setNotifySelChange(this, (NotifyProc)&C_dlg_Pos::lbChanged);

lb_bhour = new zComboBox(this, ID_POS_BHOUR, &_bhour);
lb_bhour->addCharStrings(hours);
lb_bhour->setToDefault();

lb_bmin = new zComboBox(this, ID_POS_BMIN, &_bmin);
lb_bmin->addCharStrings(minutes);
lb_bmin->setToDefault();
```

```
    lb_res = new zComboBox(this, ID_POS_RES, &_res);
    lb_res->addCharStrings(resolution);
    lb_res->setToDefault();

    local = new zRadioGroup(this, ID_POS_LOCAL, ID_POS_ZULU, &_local);
    local->setNotifyClicked(this, (NotifyProc)&C_dlg_Pos::lbChanged);

    show();
    modal();
}

int C_dlg_Pos::lbChanged(zEvent *ce)
    {
    _month = lb_month->selection() + 1;
    _day   = lb_day->selection();
    _year  = lb_year->selection() + 1993;
    switch (_month)
        {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
            {
            lb_day->reset();
            lb_day->addCharStrings(days31);
            lb_day->setToDefault();
            break;
            };
        case 4:
        case 6:
        case 9:
        case 11:
            {
            lb_day->reset();
            lb_day->addCharStrings(days30);
            lb_day->setToDefault();
            break;
            }
        case 2:
            {
            lb_day->reset();
            if ( float(_year / 4) == (_year / 4.0) &&
                                                                    float(_year / 400)
!= (_year / 400.0) )
                {
                lb_day->addCharStrings(days29);
                }
            else
                {
                lb_day->addCharStrings(days28);
```

```
                }
            lb_day->setToDefault();
            }
        }
}

int C_dlg_Pos::doDone(zEvent *ce)
    {
    storeData();
    if (completed())
        {
        fast_algorithm *alg = new fast_algorithm;
        _location = "                    ";
        lb_desc->getEditText(_location);
        node *desired = ll->find(_location);
        switch (lb_res->selection())
            {
            case 0:
                {
                delta = 1;
                break;
                }
            case 1:
                {
                delta = 5;
                break;
                }
            case 2:
                {
                delta = 10;
                break;
                }
            case 3:
                {
                delta = 15;
                break;
                }
            }

        if (desired)
            {
            _latmin = desired->data->latmin;
            _latdeg = desired->data->latdeg;
            _lonmin = desired->data->lonmin;
            _londeg = desired->data->londeg;
            _GMT    = desired->data->GMT;
            _DST    = desired->data->DST;
            _E      = desired->data->EW;
            _N      = desired->data->NS;
            }
        lo = _londeg + _lonmin / 60.0;
        if (_E) lo *= -1.0;
        f = _latdeg + _latmin / 60;
```

```
        if (_N) f *= -1.0;
        if (_local) z = TRUE;
        else z = FALSE;

        //rtn_data->lo = lo;
        //rtn_data->f = f;

        lb_month->getEditText(monthstr);
        lb_day->getEditText(daystr);
        lb_year->getEditText(yearstr);
/*      strcpy(rtn_data->array[0],"***** S U N &  M O O N  P O S I T I O N  C H A R T *****");
        rtn_data->array[0] = strdup(temp);
        temp = NULL;
        strcat(temp, _location);
        strcat(temp,(_N ? "  N " : "  S "));
        itoa(_latdeg, trash, 10);
        strcat(temp, trash);
        strcat(temp, " ");
        gcvt(_latmin, 5, trash);
        strcat(temp, trash);
        strcat(temp,(_E ? "  E " : "  W ") );
        itoa(_londeg, trash, 10);
        strcat(temp, trash);
        strcat(temp, " ");
        gcvt(_lonmin, 5, trash);
        strcat(temp, trash);
        strcpy(rtn_data->array[1],temp);
        temp = monthstr;
        strcat(temp, " ");
        strcat(temp, daystr);
        strcat(temp, " ");
        strcat(temp, yearstr);
        strcat(temp, "   Zulu + ");
        gcvt(_GMT, 5, trash);
        strcat(temp, trash);
        strcpy(rtn_data->array[2],temp);
        strcpy(rtn_data->array[5],"       Sun     Sun     Lux     Moon     Moon     %Moon     Lux");
        strcpy(rtn_data->array[7],"Time  Azimuth  Altitude  Illum  Azimuth  Altitude (Phase)  Illum");
        strcpy(rtn_data->array[8],"-------------------------------------------------------------");
        //logic for producing times
        for (count = 0; count <=60; count++, _bmin += delta)
            {
            if (_bmin >= 60)
                {
                _bmin -=60;
                _bhour++;
                }
            h = _bhour * 100 + _bmin;
            alg->illum(lo, f, _year+1993, _month+1, _day+1, z, h, _saz, _sal, _sil, _maz, _mal, _mil, _mpr,
_til);
            if (h >= 2400) h -= 2400;
            sprintf(temp, "%04.0f %8.0f %9.0f %13.5f %8.0f %9.0f %9.0f %12.5f", h, _saz, _sal, _sil,
_maz, _mal, _mpr, _til);
```

```
                strcpy(rtn_data->array[count+8],temp);
                }
*/      }
    shutdown();
    return 1;
    }


/*POS_CHART_DLG::POS_CHART_DLG(zResId& rid) : zFormDialog(app->rootWindow(),rid)
        {
        display_array = new zComboBox(this, POS_CHART_ARRAY);
    for (int count = 0; count = 80; count++)
                {
                display_array->add(array[count][0]);
                }
        }
*/
```

FASTAL.CPP

```
/*************************************************************************/
/* File Name:  fastal.cpp
/* Programmer:  Michael T. Lester
/* Created:    05 Nov 1993          Modified: 21 Feb 1994
/* Description: Implementation of algorithms from Circular 171
/* Compiler:   Borland C++ (DOS IDE)
/*************************************************************************/

#include <math.h>
#include "fastal.h"

#define sgn(arg) (arg<0 ? (-1) : (1))
#define deg(arg) (floor(arg) + ((arg - floor(arg))*10)/6)
#define dms(arg) (floor(arg) + 6 * (arg - floor(arg))/10)
#define neg56 (-5/6)
#define TRUE  1
#define FALSE 0

void
fast_algorithm::illum(
                        float     LO,     //Longitude, degrees, E=+, W=-
                        float     F,              //Latitude, degrees, N=+, E=-
                        float     IY,     //Year (####)
                        int       IM,     //Month (01 - 12)
                        int       ID,     //Day (01 - 31) assumed correct for calendar
                        float     Z,              //Time format, 0=Zulu, 1=local (zone)
                        float     H,              //Hour, 24 hr format, zulu or local as above
                        float&    saz,  //result, Sun Azimuth
                        float&    sal,    //result, Sun Altitude
                        float&    sil,    //result, Sun Illuminance
                        float&    maz,    //result, Moon Azimuth
                        float&    mal,    //result, Moon Altitude
                        float&    mil,    //result, Moon Illuminance
                        float&    mpr,    //result, Moon Percentage of Full
                        float&    til     //result, Total Illuminance
                        ){

    RD  = 57.29578;
    DR  = 1 / RD;
    CE  = 0.91775;
    SE  = 0.39715;
    A[0] = -0.01454;
    A[1] = -0.010453;
    A[2] = -0.020791;
    A[3] = 0.00233;

    F  = F * DR;
    C  = 360;
    LI = fabs(LO);
    SI = sin(F);
```

```
    CI = cos(F);
    J  = 367 * IY - floor(7 * (IY + floor((IM + 9) / 12)) / 4) +
                                                                    floor(275
* IM / 9) + ID - 730531;
    DT = 0;

    if (Z == 0) DT = -LO / C;
    if (Z == 1) DT = -(LI - 15 * floor((LI + 7.5) / 15)) / C * sgn(LO);

    Z0 = J -0.5;
    E  = (deg(H / 100.0)) / 24 - DT - LO / 360;
    D  = Z0 + E;
    TD = 280.46 +0.98565 * D;
    T  = TD - floor(TD / 360) * 360;

    if (T < 1e-500) T = T + 360;

    TD  = 357.5 +0.9856 * D;
    G   = (TD - floor(TD / 360) * 360) * DR;
    LS  = (T + 1.91 * sin(G)) * DR;
    AS1 = atan(CE * tan(LS)) * RD;
    Y   = cos(LS);

    if (Y < 1e-500) AS1 = AS1 + 180;

    SD = SE * sin(LS);
    DS = asin(SD);
    T  = T - 180;
    T  = T + 360 * E + LO;

    for (int N = 1; N <3; N++){
        if (N == 2){
            TD = 218.32 + 13.1764 * D;
            V  = TD - floor(TD / 360) * 360;

            if (V < 1e-500) V = V + 360;

            TD  = 134.96 + 13.06499 * D;
            Y   = (TD - floor(TD / 360) * 360) * DR;
            TD  = 93.27 + 13.22935 * D;
            O   = (TD - floor(TD / 360) * 360) * DR;
            TD  = 235.7 + 24.3815 * D;
            W   = (TD - floor(TD / 360) * 360) * DR;
            SB  = sin(Y);
            CB  = cos(Y);
            X   = sin(O);
            S   = cos(O);
            SD  = sin(W);
            CD  = cos(W);
            V   = V + (6.29 - 1.27 * CD +0.43 * CB) * SB + (.66 + 1.27 * CB) *

    SD -0.19 * sin(G) -0.23 * X * S;
            V   = V * DR;
```

```
        Y  = ((5.13 -0.17 * CD) * X + (.56 * SB +0.17 * SD) * S) * DR;
        SV = sin(V);
        SB = sin(Y);
        CB = cos(Y);
        Q  = CB * cos(V);
        P  = CE * SV * CB - SE * CB;
        SD = SE * SV * CB + CE
        AS1 = atan(P / Q) * RD;

        if (Q < 0) AS1 = AS1 + 180;

        DS = asin(SD);
    }

    H = T - AS1;
    CD = cos(DS);
    CS = cos(H * DR);
    Q = SD * CI - CD * SI * CS;
    P = -CD * sin(H * DR);
    AZ = atan(P / Q) * RD;

    if (Q < 1e-500) AZ = AZ + 180;
    if (AZ < 1e-500) AZ = AZ + 360;

    AZ = floor(AZ +0.5);
    H = asin(SD * SI + CD * CI * CS) * RD;
    Z = H * DR;
    H = H -0.95 * (N - 1) * cos(H * DR);
    HA = H;

    if (H >= (-5 / 6)){
        HA = H + 1 / (tan((H + 8.59 / (H + 4.42)) * DR)) / 60;
    }

    U = sin(HA * DR);
    X = 753.6616;
    S = asin(X * cos(HA * DR) / (X + 1));
    M = X * (cos(S) - U) + cos(S);
    M = exp(-.21 * M) * U +0.0289 * exp(-.042 * M) *

(1 + (HA + 90) * U / 57.29578);
    HA = ((floor(fabs(HA) +0.5)) * sgn(HA));

    if (N == 1){
        IS1 = 133775 * M;
        saz = AZ;
        sal = HA;
        sil = IS1;
    }

    else{

        E  = acos(cos(V - LS) * CB);
```

```
            P  = 0.882 * exp(-3.343 / (pow( tan(E / 2), 0.632))) + 0.0344 *

            (sin(E) - E * cos(E));
            P  = 0.418 * P / (1 -0.005 * cos(E) -0.03 * sin(Z));
            IL = P * M;
            IS1 = IS1 + IL +0.0005;

            maz = AZ;
            mal = HA;
            mil = IL;
            IL = floor(50 * (1 - cos(E)) +0.5);
            mpr = IL;
            til = mil + sil;
        }
    }
}

void
fast_algorithm::event( float        LO,     //Longitude, degrees, E=+, W=-
                       float        F,              //Latitude, degrees, N=+, S=-
                       float        IY,     //Year (####)
                       int          IM,     //Month (01 - 12)
                       int          ID,     //Day (01 - 31) assumed correct for calendar
                       float        Z,              //Time format, 0=Zulu, 1=local (zone)
                       float        H,      //hours from zulu
                       float&       sr,     //result, Sun Rise
                       float&       sst,    //result, Sun Set Time
                       float&       st,     //result, Time of Sun Meridian Passage
                       float&       sa,     //result, Altitude of Sun Meridian Passage
                       float&       mr,     //result, Moon Rise
                       float&       ms,     //resutl, Moon Set
                       float&       mt,     //result, Time of Moon Meridian Passage
                       float&       ma,     //result, Altitude of Moon Meridian Passage
                       float&       ac,     //result, Morning Civil Twilight
                       float&       an,     //result, Morning Nautical Twilight
                       float&       pc,     //result, Evening Civil Twilight
                       float&       pn,     //result, Evening Nautical Twilight
                       float&       dl              //result, length of daylight
                       )
    {
    RD  = 57.29578;
    DR  = 1 / RD;
    CE  = 0.91775;
    SE  = 0.39715;
    A[0] = -0.01454;
    A[1] = -0.10453;
    A[2] = -0.20791;
    A[3] = 0.00233;
    int   exit_now = FALSE, exit_n_loop = FALSE;

    F = F * DR;
    C = 360;
    LI = fabs(LO);
```

```
    SI = sin(F);
    CI = cos(F);
    J  = 367 * IY - floor(7 * (IY + floor((IM + 9) / 12)) / 4) +
                                                                    floor(275
* IM / 9) + ID - 730531;
    DT = 0;

    if (!Z) DT = -LO / C;
    if (Z == 1) DT = -(LI - 15 * floor((LI + 7.5) / 15)) / C * sgn(LO);
    Z0 = J -0.5;
    for(int L = 1; L <= 4; L++){
        if (L == 4) C = 347.81;
        if ((L == 1) || (L == 4)){
            M = 0.5 + DT;
            K = 1;
            while (1) {
                exit_now = FALSE;
                M -= DT;
                E = M - LO / 360;
                D = Z0 + E;
                if (fabs(E) >= 1) E -= sgn(E);
                TD = 280.46 + .98565 * D;
                T = TD - floor(TD / 360) * 360;
                if (T < 0) T += 360;
                TD = 357.5 + .9856 * D;
                G = (TD - floor(TD / 360) * 360) * DR;
                LS = (T + 1.91 * sin(G)) * DR;
                AS1 = atan(CE * tan(LS)) * RD;
                Y = cos(LS);
                if (Y < 0) AS1 += 180;
                SD = SE * sin(LS);
                DS = asin(SD);
                T -= 180;
                if (L == 4){
                    TD = 218.32 + 13.1764 * D;
                    V = TD - floor(TD / 360) * 360;
                    if (V < 0) V = V + 360;
                    TD = 134.96 + 13.06499 * D;
                    Y = (TD - floor(TD / 360) * 360) * DR;
                    TD = 93.27 + 13.22935 * D;
                    O = (TD - floor(TD / 360) * 360) * DR;
                    TD = 235.7 + 24.3815 * D;
                    W = (TD - floor(TD / 360) * 360) * DR;
                    SB = sin(Y);
                    CB = cos(Y);
                    X = sin(O);
                    S = cos(O);
                    SD = sin(W);
                    CD = cos(W);
                    V += (6.29 - 1.27 * CD + .43 * CB) * SB + (.66 + 1.27 * CB) *

    SD - .19 * sin(G) - .23 * X * S;
                    V *= DR;
```

```
        Y = ((5.13 - .17 * CD) * X + (.56 * SB + .17 * SD) * S) * DR;
        SV = sin(V);
        SB = sin(Y);
        CB = cos(Y);
        Q = CB * cos(V);
        P = CE * SV * CB - SE * SB;
        SD = SE * SV * CB + CE * SB;
        AS1 = atan(P / Q) * RD;
        if (Q < 0) AS1 += 180;
        DS = asin(SD);
    }
    T += LO + 360 * E;
    T -= floor(T / 360) * 360;
    U = T - AS1;
    if (fabs(U) > 180) U -= 360 * sgn(U);
    U /= C;
    M += DT - U;
    if (L < 4) K++;
    switch (K){

        case 1:
        case 3:
        case 5:{
            K++;
            break;
        }

        case 2:{
            if( (M>=0) && (M < 1)){
                    exit_now = TRUE;
            }
            else{
                    M -= sgn(M);
                    K++;
            }
            break;
        }

        case 4:{
            if (M >= 0){
                    exit_now = TRUE;
                    break;
            }
            M -= sgn(M);
            K++;
            break;
        }

        case 6:{
            exit_now = TRUE;
            break;
        }
    } //switch
```

```
                if (exit_now) break;
            } // while loop
            H = asin(cos(F - DS)) * RD;
            if (L == 4) H -= 0.95 * cos(H);
            HA = H;
            if (H >= (neg56))
                        HA = H + 1 / (tan((H + 8.59 / (H + 4.42)) * DR)) / 60;
        } // if ((L == 1) || (L == 4)
        H = (A[L-1] - SI * SD) / (CI * cos(DS));
        if (fabs(H) > 1) H = 1.5;
        else H = acos(H) * RD / C;
        B[0] = M - H;
        B[1] = M + H;
        for(int I = 0; I<=1; I++){
            K = 2 * (I + 1) - 3;
            for (int N = 1; N <= 6; N++){
                exit_n_loop = FALSE;
                B[I] -= DT;
                E = B[I] -LO / 360;
                D = Z0 + E;
                if (fabs(E) >= 1) E -= sgn(E);
                TD = 280.46 + .98565 * D;
                T = TD - floor(TD / 360) * 360;
                if (T < 0) T += 360;
                TD = 357.5 + .9856 * D;
                G = (TD - floor(TD / 360) * 360) * DR;
                LS = (T + 1.91 * sin(G)) * DR;
                AS1 = atan(CE * tan(LS)) * RD;
                Y = cos(LS);
                if (Y < 0) AS1 += 180;
                SD = SE * sin(LS);
                DS = asin(SD);
                T -= 180;
                if (L == 4){
                    TD = 218.32 + 13.1764 * D;
                    V = TD - floor(TD / 360) * 360;
                    if (V < 0) V += 360;
                    TD = 134.96 + 13.06499 * D;
                    Y = (TD - floor(TD / 360) * 360) * DR;
                    TD = 93.27 + 13.22935 * D;
                    O = (TD - floor(TD / 360) * 360) * DR;
                    TD = 235.7 + 24.3815 * D;
                    W = (TD - floor(TD / 360) * 360) * DR;
                    SB = sin(Y);
                    CB = cos(Y);
                    X = sin(O);
                    S = cos(O);
                    SD = sin(W);
                    CD = cos(W);
                    V += (6.29 - 1.27 * CD + .43 * CB) * SB + (.66 + 1.27 * CB) *

SD - .19 * sin(G) - .23 * X * S;
                    V *= DR;
```

```
                Y = ((5.13 - .17 * CD) * X + (.56 * SB + .17 * SD) * S) * DR;
                SV = sin(V);
                SB = sin(Y);
                CB = cos(Y);
                Q = CB * cos(V);
                P = CE * SV * CB - SE * SB;
                SD = SE * SV * CB + CE * SB;
                AS1 = atan(P / Q) * RD;
                if (Q < 0) AS1 += 180;
                DS = asin(SD);
            } //if (L == 4)
            T += LO + 360 * E;
            T -= floor(T / 360) * 360;
            U = T - AS1;
            if (fabs(U) > 180) U -= 360 * sgn(U);
            U /= C;
            H = (A[L-1] - SI * SD) / (CI * cos(DS));
            if (fabs(H) > 1) H = 1.5;
            else H = acos(H) * RD / C;
            B[I] += K * H - U + DT;
            if (L < 4) N++;
            switch (N){
                case 1:
                case 3:
                case 5:{
                    break;
                }

                case 2:{;
                    if ( (B[I] >= 0) && (B[I] < 1)){
                            exit_n_loop = TRUE;
                    }
                    else B[I] -= sgn(B[I]);
                    break;
                }

                case 4:{
                    if (B[I] >= 0){
                            exit_n_loop = TRUE;
                    }
                    break;
                }

                case 6:{
                    exit_n_loop = TRUE;
                    break;
                }
            } //switch
            if (exit_n_loop) break;
        } //N for loop
    } //I for loop
    switch (L){
        case 1:{
```

```
        R = floor(100 * dms(M * 24) + .5);
        st = R;
        HA = floor( fabs(HA) + .5) * sgn(HA);
        sa = HA;
        for (int I = 0; I <= 1; I++){
            R = floor( 100 * dms(B[I] * 24) + .5);
            if ( (R >= 4800) || (R < 0) ) break;
            if (!I){
                sr = R;
                continue;
            }
            else {
                sst = R;
            }
            R = B[1] - B[0];
            if (R < 0) R++;
            R = floor( 100 * dms(R * 24) + .5);
            dl = R;
        } //I for loop
break;
} // case 1
        case 2:{
        for (int I = 0; I <= 1; I++){
            R = floor( 100 * dms(B[I] * 24) + .5);
            if ( (R >= 4800) || (R < 0) ) break;
            if (!I)
                ac = R;
            else
                pc = R;
        } // for loop
break;
}// case 2

        case 3: {
        for (int I = 0; I <= 1; I++){
            R = floor( 100 * dms(B[I] * 24) + .5);
            if ( (R >= 4800) || (R < 0) ) break;
            if (!I)
                an = R;
            else
                pn = R;
        } // for loop
break;
} // case 3

        case 4: {
        R = floor( 100 * dms(M * 24) + .5);
        mt = R;
        HA = floor( fabs(HA) + .5) * sgn(HA);
        ma = HA;
        for (int I = 0; I <= 1; I++){
            R = floor( 100 * dms(B[I] * 24) + .5);
            if ( (R >= 4800) || (R < 0) ) break;
```

```
                        if (ll)
                            mr = R;
                        else
                            ms = R;
                    } // for loop
                } // case 4
            } // Switch
        } // L loop
} //main


/*
char* fast_algorithm::calendar(int b_day, int b_mo, int b_yr,
                                                    int e_day, int e_mo, int e_yr,
                                                float lat, float lon, int Z,
                                                float thresh, int   delta){

//      int days;     //# of days data is requested for
        int JB, JE;  //Julian for Begin and End of requested period respectively
        char *matrix; // pointer to the calendar matrix

        // compute # of days requested
        JB = 367 * b_yr - floor(7 * (b_yr + floor((b_mo + 9) / 12)) / 4) +
                                                                    floor(275 * b_mo / 9) +
b_day - 730531;
        JE = 367 * e_yr - floor(7 * (e_yr + floor((e_mo + 9) / 12)) / 4) +
                                                                    floor(275 * e_mo / 9) +
e_day - 730531;
        const int days = JE - JB + 11;

        // allocate space for array and fill in header and footer
        matrix = new char[132][60];
//      matrix[47][0] = "LIGHT LEVEL PLANNING CALENDAR";
//      for(int count=0; count <=131; count++) matrix[count][4] = "_";


        return(matrix);
}
*/
```

LL.CPP

```
//
// ll.cpp
//
// routines for ll.h, linked list of location data
//
#include "ll.h"
#include "dialogs.h"
#include <string.h>

    //types of files used in the Save_as dialog
char *types[] =
    {
    "Data Files (*.dat)", "*.dat",
    "All Files (*.*)", "*.*",
    0,0
    };

    //Initialize a new node of the linked list of locations
node::node()
    {
    data = NULL;
    next = NULL;
    }

    //Destructor for a node of the linked list of locations
node::~node()
    {
    delete data;
    }

    //Constructor for the linked list of locations
locations_list::locations_list()
    {
    datafile = "moonlite.dat";
    list_dirty = FALSE;
    head = NULL;
    }

locations_list::clear()
    {
    if (list_dirty)
        {
        zMessage *msg = new zMessage(app->rootWindow(), "Locations have Changed!\nSave
Changes?",
                                     "WARNING",
                                     MB_YESNOCANCEL | MB_ICONSTOP);
        if (msg->value() == IDCANCEL)
            {
            return (0);
            }
```

```
        if (msg->value() == IDYES)
            {
            save();
            }
        }
    node *temp, *temp1;                     //temp pointers
    temp = head;                            //point to first node
    head = NULL;                                    //clear head pointer
    while (temp != NULL)                    //step through list, deleting each node
        {                                                   //the node deletes the data portion
        temp1 = temp->next;
        delete temp;
        temp = temp1;
        }
    list_dirty = FALSE;                     //set list as clean
    return (1);             //return ok.
    }

locations_list::open_dialog(zWindow *pwin)
    {
    zFileOpenForm *fs = new zFileOpenForm(app->rootWindow(), "Open File",
                                                    (char *) datafile, types);
    if (fs->completed())        //if dialog was completed ok
        {
        datafile = fs->name();              // use the new file name
        open();                                     // and open that file.
        }
        return (1);
    }

locations_list::open()
    {
    clear();                //clear the old list, to check if clean
    FILE *fp;
    node *temp;
    int result;
    fp  = fopen(datafile, "rb");    //open the file
    if (!fp) return(0);             //open fails, report same
    while(1)                        //if opened ok, continue
        {
        temp = new node;
        temp->data = new location_struct;
        result = fread(temp->data, sizeof(location_struct), 1, fp);  //read data
        if (result == 0)            //if data read failed
            {                       // delete temp node, and return
            delete temp;
            break;
            }
        temp->next = head;
        head = temp;
        }
    if (fclose(fp) != 0) return(0);  //if close fails, return failed
    list_dirty = FALSE;             //set list to clean
```

```
    return(1);
    }

int
locations_list::save()
    {
    node *temp;
    FILE *fp;
    fp = fopen(datafile, "wb");
    if (!fp) return (0);
    temp = head;
    while (temp != NULL)
        {
        fwrite(temp->data, sizeof(location_struct), 1, fp);
        temp = temp->next;
        }
    fclose(fp);
    list_dirty = FALSE;
    return (1);
    }

int
locations_list::saveas(zWindow *pwin)
    {
    zFileSaveAsForm *fs = new zFileSaveAsForm(app->rootWindow(), "Save As",
                                                (char *) datafile, types);
    if (fs->completed())
        {
        datafile = fs->name();
        save();
        }
        return (1);
    }

locations_list::add_record(location_struct *temp_location)
    {
    node *temp_node, temp;
    temp_node = new node;
        //point node to data structure and link node into list
    temp_node->data = temp_location;
    temp_node->next = head;
    head = temp_node;
    list_dirty = TRUE;
    return (1);
    }

locations_list::add_dialog(zWindow *pwin, locations_list *ll)
    {
    C_dlg_locations *dlg_locs = new C_dlg_locations(ll, pwin,
                                    zResId(DIALOG_LOCATIONS));
    if(dlg_locs->completed())
        {
            //make some temporary pointers
```

```
        location_struct *temp_location;
            //make a new node and new data structure
        temp_location = new location_struct;
            //move data from calling procedure to data structure
        strcpy(temp_location->desc, dlg_locs->location());
        temp_location->latdeg   = dlg_locs->_latdeg;
        temp_location->latmin   = dlg_locs->_latmin;
        temp_location->NS       = dlg_locs->_N - 305;
        temp_location->londeg = dlg_locs->_londeg;
        temp_location->lonmin   = dlg_locs->_lonmin;
        temp_location->EW       = dlg_locs->_E - 309;
        temp_location->GMT = dlg_locs->_GMT;
        temp_location->DST = dlg_locs->_DST;

        add_record(temp_location);
        delete dlg_locs;
        }
        //if all was successful, return true
    return(TRUE);
    }

locations_list::edit_dialog(zWindow *pwin, locations_list *ll)
    {
    C_dlg_loc_edit *dlg_edit = new C_dlg_loc_edit(ll, pwin, zResId(EDIT_LOCATION), head);
        //make some temporary pointers
    node *temp = head;
    if (dlg_edit->completed())
        {
        while ((temp != NULL) && (temp->data->desc != dlg_edit->_location))
            {
            temp = temp->next;
            }
            if (temp != NULL)
                {
                strcpy(temp->data->desc, dlg_edit->location());
                temp->data->latdeg      = dlg_edit->_latdeg;
                temp->data->latmin      = dlg_edit->_latmin;
                temp->data->NS                  = dlg_edit->_N - 505;
                temp->data->londeg      = dlg_edit->_londeg;
                temp->data->lonmin      = dlg_edit->_lonmin;
                temp->data->EW                  = dlg_edit->_E - 509;
                temp->data->GMT                 = dlg_edit->_GMT;
                temp->data->DST                 = dlg_edit->_DST;
                list_dirty = TRUE;
                }
        }
    delete dlg_edit;

        //If all was successful, return true
    return(TRUE);
    }
```

```
locations_list::delete_dialog(zWindow *pwin, locations_list *ll)
    {
    C_dlg_loc_del *dlg_del = new C_dlg_loc_del(ll, pwin, zResId(LOCATION_DELETE), head);
        //make some temporary pointers
    node *temp, *temp1;
    temp = head;
    temp1 = temp;
    if (dlg_del->completed())
        {
        while ((temp != NULL) && (temp->data->desc != dlg_del->_location))
            {
            temp1 = temp;
            temp = temp->next;
            }
        if (temp != NULL)
            {
            if (temp == head)
                {
                head = temp->next;
                }
            else
                {
                temp1->next = temp->next;
                }
            delete temp;
            list_dirty = TRUE;
            }
        }
    delete dlg_del;

        //if all was successful, return true
    return(TRUE);
    }

node* locations_list::find(zString &desc)
    {
     node *temp;
     temp = head;
     while (temp != NULL)
        {
        if (temp->data->desc == desc)
            {
            return (temp);
            }
        temp = temp->next;
        }
     return (NULL);
     }

char*
locations_list::get_next()
    {
     node *old = temp_ptr;
```

```
if (temp_ptr) temp_ptr = temp_ptr->next;
if (old) return (old->data->desc);
else return (NULL);
}
```

LOCATIONS.CPP

```
#include<stdio.h>
#include<stdlib.h>
#include<iostream.h>
#include<string.h>
#include "location.h"

node::node()
{
    next = NULL;
}

locations::locations()
{
    head = NULL;
    f_name = "zzzz.dat";
}


int
locations::add()
{
    node *temp;
    char *string = "string here";
    float lat = 123.45, lng = 678.90;
    float GMT = 8.0;
    int   DST = 1, lock;

    for (int x = 0; x<3; x++)
    {
        cout << "\nenter a number...";
        cin >> lock;
        temp = new node;
        strncpy(temp->data.desc, string, DESC_LENGTH);
        //temp->data.desc = string;
        temp->data.latitude = lat;
        temp->data.longitude = lng;
        temp->data.GMT = GMT;
        temp->data.DST = DST;
        temp->data.lock = lock;
        temp->next = head;
        head = temp;
    }
    return(1);
}

int
locations::save()
{
    FILE *fp;
    node *temp;
```

```
    fp  = fopen(f_name, "wb");
    if (!fp) return(0);
    temp = head;
    while (temp != NULL)
        {
        fwrite(&temp->data, sizeof(S_location), 1, fp);
        temp = temp->next;
        }
    if (fclose(fp) != 0) return(0);
    return(1);
}

int
locations::load()
{
    FILE *fp;
    node *temp;
    int result;
    fp  = fopen(f_name, "rb");
    if (!fp) return(0);
    while(1)
        {
        temp = new node;
        result = fread(&temp->data, sizeof(S_location), 1, fp);
```

```
            if (result == 0)
                {
                delete temp;
                break;
                }
            temp->next = head;
            head = temp;
            }
        if (fclose(fp) != 0) return(0);
        return(1);
}
```

MENUFRAM.CPP

```
//
// menufram.cpp
//
#include "menufram.h"

MenuFrame::MenuFrame(zWindow* parent,zSizer* siz,DWORD winStyle, const char*
title):zAppFrame(parent,siz,winStyle,title) {
    menu(new zMenu(this, zResId(MENU_MAIN)));
    menu()->setCommand(this,(CommandProc)&MenuFrame::doExit,ID_MENU_FILEEXIT);
    menu()->setCommand(this,(CommandProc)&MenuFrame::AddLocation,ID_MENU_LOCS_ADD);
}

MenuFrame::~MenuFrame() {
    NULL;
}

int
MenuFrame::doExit(zCommandEvt *ce)
{
    zMessage mess(app->rootWindow()," Exit MOONLITE?","",MB_OKCANCEL);
    if (mess.isOk()) app->quit();
    return 1;
}

MenuFrame::AddLocation(zCommandEvt *ce)
{
    ll->add(this);
}

MenuFrame::command(zCommandEvt *ce) {
    tp->clearRect();
    tp->moveTo(0,0);
    return 1;
}
```

MOONLITE.CPP

```
//
//    MOONLITE.cpp
//
//
#include <stdlib.h>
#include "bmpshow.h"
#include "zapp.hpp"
#include "defines.h"
#include "ll.h"
#include "dialogs.h"
#include "routines.h"

locations_list *ll = new locations_list;

class MenuFrame : public zAppFrame{

     private:

          BmpShowPane *bmpPane;
          zTextPane* tp;

     public:

          MenuFrame(zWindow* parent,zSizer* siz,
                                                                    DWORD
winStyle,const char* title);
          ~MenuFrame();
          virtual int command(zCommandEvt *);
          int doExit(zCommandEvt *);
          int AddLocation(zCommandEvt *);
          int DeleteLocation(zCommandEvt *);
          int EditLocation(zCommandEvt *);
          int FileSave(zCommandEvt *);
          int FileOpen(zCommandEvt *);
          int FileNew(zCommandEvt *);
          int FileSaveAs(zCommandEvt *);
          int SpotData(zCommandEvt *);
          int Event(zCommandEvt *);
          int Position(zCommandEvt *);
          int kill(zEvent *);
          doOk();
};

MenuFrame::MenuFrame(zWindow* parent,zSizer* siz,DWORD winStyle, const char*
title):zAppFrame(parent,siz,winStyle,title) {
     bmpPane = new BmpShowPane(this, new zSizeWithParent);
     bmpPane->show();
     FILE *fp;
     fp = fopen("moonlite.bmp","r");     //open the file
     if (fp)
```

```
        {
        fclose(fp);
        bmpPane->display("moonlite.bmp");                    //open ok, display bmp
        }
    setIcon(new zIcon(zResId(ICON_1)));
    menu(new zMenu(this, zResId(MENU_MAIN)));
    menu()->setCommand(this,(CommandProc)&MenuFrame::doExit,ID_MENU_FILEEXIT);
    menu()->setCommand(this,(CommandProc)&MenuFrame::AddLocation,ID_MENU_LOCS_ADD);
    menu()->setCommand(this,(CommandProc)&MenuFrame::EditLocation,ID_MENU_LOCS_EDIT);
    menu()-
>setCommand(this,(CommandProc)&MenuFrame::DeleteLocation,ID_MENU_LOCS_DELETE);
    menu()->setCommand(this,(CommandProc)&MenuFrame::FileSave,ID_MENU_FILESAVE);
    menu()->setCommand(this,(CommandProc)&MenuFrame::FileOpen,ID_MENU_FILEOPEN);
    menu()->setCommand(this,(CommandProc)&MenuFrame::FileNew,ID_MENU_FILENEW);
    menu()->setCommand(this,(CommandProc)&MenuFrame::FileSaveAs,ID_MENU_FILESAVEAS);
    menu()->setCommand(this,(CommandProc)&MenuFrame::SpotData,ID_MENU_SPOTDATA);
    menu()->setCommand(this,(CommandProc)&MenuFrame::Event,ID_MENU_DAILYEVENTS);
    menu()->setCommand(this,(CommandProc)&MenuFrame::Position,ID_MENU_POSITION);
}

MenuFrame::~MenuFrame() {
    NULL;
}

int
MenuFrame::doExit(zCommandEvt *ce)
{
    if (ll->peek_list_dirty())
        {
        zMessage *msg = new zMessage(this, "Locations have Changed!\nSave Changes?",
                                     "W A R N I N G",
                                     MB_YESNO | MB_ICONSTOP);
        if (msg->value() == IDYES)
            {
            ll->save();
            }
        }
    app->quit();
}

MenuFrame::AddLocation(zCommandEvt *ce)
{
    ll->add_dialog(this, ll);
}

MenuFrame::EditLocation(zCommandEvt *ce)
{
    ll->edit_dialog(this, ll);
}

MenuFrame::DeleteLocation(zCommandEvt *ce)
{
    ll->delete_dialog(this, ll);
```

```
}

MenuFrame::FileSave(zCommandEvt *ce)
{
    ll->save();
}

MenuFrame::FileSaveAs(zCommandEvt *ce)
{
    ll->saveas(this);
}

MenuFrame::FileOpen(zCommandEvt *ce)
{
    ll->open_dialog(this);
}

MenuFrame::FileNew(zCommandEvt *ce)
{
    ll->clear();
    return(1);
}

MenuFrame::SpotData(zCommandEvt *ce)
{
    routines *engine = new routines(ll);
    engine->spotdata();
    delete engine;
    return(1);
}

MenuFrame::Event(zCommandEvt *ce)
{
    routines *engine = new routines(ll);
    engine->event();
    delete engine;
    return(1);
}

MenuFrame::Position(zCommandEvt *ce)
{
    routines *engine = new routines(ll);
    engine->position();
    delete engine;
    return(1);
}

MenuFrame::command(zCommandEvt *ce) {
    tp->clearRect();
    tp->moveTo(0,0);
    return 1;
}
```

```
int
MenuFrame::kill(zEvent *ev)
    {
    return ( ll->clear() );
    }

void zApp::main() {
    MenuFrame *mainWnd=new MenuFrame(0,new zSizer(10,10,625,520),zSTDFRAME,"MOONLITE");
    mainWnd->show();
    if (!(ll->open()))
        {
        zMessage *msg = new zMessage(app->rootWindow(), "MOONLITE.dat not found.\nNo locations
loaded",
                                                "W A R N I N G",
                                                MB_OK | MB_ICONSTOP);
        }
    go();
    delete mainWnd;
}
```

ROUTINES.CPP

```
//
// routines.cpp
//

#include<alloc.h>
#include "routines.h"
#include "dialogs.h"
#include "ll.h"

routines::routines(locations_list *list_in)
    {
    ll = list_in;
    }


routines::spotdata()
    {
    C_dlg_spotdata *dlg_spot = new C_dlg_spotdata(ll, zResId(SPOT_DETAIL));
    delete dlg_spot;
    }

routines::event()
    {
    C_dlg_Event *dlg_event = new C_dlg_Event(ll, zResId(EVENTS));
    delete dlg_event;
    }

routines::position()
    {
    C_dlg_Pos *pos_get = new C_dlg_Pos(ll, zResId(POSITION));
    delete pos_get;
//  POS_CHART_DLG *pos_chart = new POS_CHART_DLG(zResId(Position_Chart));
//  delete pos_chart;
    }
```

LIST OF REFERENCES

[1] United States Marine Corps, Marine Aviation Weapons and Tactics Squadron One, *Helicopter NVG Manual*, Assault Support Division, MAWTS-1, Yuma, Arizona 85369-6073

[2] United States Marine Corps, Marine Aviation Weapons and Tactics Squadron One, *Litelevl User's Guide*, MAWTS-1, Yuma, Arizona 85369-6073

[3] Conversations between Dr. P. M. Janiczek, United States Naval Observatory, Astronomical Applications Division and the author, 1992 - 1994

[4] Personal notes of Dr. P.M. Janiczek used for the creation of reference [2].

[5] United States Naval Observatory Circular No. 171, *Computer Programs for Sun and Moon Illuminance With Contingent Tables and Diagrams*, P.M. Janiczek and J.A. DeYoung, February 19, 1987.

[6] Borland International, Borland C++ Version 4.0 Programmers Guide, 1993

[7] Hennessy, John L, and Patterson, David A, *Computer Architecture, a Quantitative Approach*, Morgan Kaufmann, 1990.

[8] Inmark Development, *Zapp Programmer's Reference*, 1993

[9] Handbook of Geophysics and the Space Environment. Scientific editor: Adolph S. Jursa. AFGL, AFSYSCOM (1985). Order from NTIS, Springfield, VA 22161, Document Accession No: ADA 167000.

[10] Stubbs, Daniel F. & Webre, Neil W., *Data Structures with Abstract Data Types and ADA*, PWS-Kent, 1993

BIBLIOGRAPHY

Berry, John Thomas, *C++ Programming*, Sams, 1992

McCord, James W., *Borland C++ Programmer's Reference 2nd Edition*, Que, 1992

Person, Ron & Rose, Karen, *Using Windows 3.1*, Que, 1993

Shammas, Namir Clement, *Advanced C++*, Sams, 1992

Stroustrup, Bjarne, *The C++ Programming Language*, Addison-Wesley, 1991

United States Congress, House of the, *Night Vision Goggles, Hearing before the Investigation Subcommittee on Armed Services*, U.S. Government Printing Office, Washington, D.C. 1989

Yourdon, Edward, *Modern Structured Analysis*, Yourdon Press, 1989

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5101	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	1
4.	Professor Douglas J. Fouts, Code EC/Fs Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	2
5.	Raymond F. Bernstein Jr., Code EC/Be Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	1
6.	Director, Training and Education MCCDC, Code C46 1019 Eliot Rd. Quantico, VA 22134-5027	1
7.	Doctor Paul M. Janiczek United States Naval Observatory 3450 Massachusets Avenue N.W. Washington, DC 20392-5420	1
8.	Mr. Jules Lewyckyj Naval Air Warfare Center Warminster, PA 18974	1

9. Captain Michael T. Lester
125 Leidig Circle
Monterey, CA 93940 3